

Universal Psychometrics Tasks: difficulty, composition and decomposition

José Hernández-Orallo

DSIC, Universitat Politècnica de València, Spain

jorallo@dsic.upv.es

March 27, 2015

Abstract

This note revisits the concepts of task and difficulty. The notion of cognitive task and its use for the evaluation of intelligent systems is still replete with issues. The view of tasks as MDP in the context of reinforcement learning has been especially useful for the formalisation of learning tasks. However, this alternate interaction does not accommodate well for some other tasks that are usual in artificial intelligence and, most especially, in animal and human evaluation. In particular, we want to have a more general account of episodes, rewards and responses, and, most especially, the computational complexity of the algorithm behind an agent solving a task. This is crucial for the determination of the difficulty of a task as the (logarithm of the) number of computational steps required to acquire an acceptable policy for the task, which includes the exploration of policies and their verification. We introduce a notion of asynchronous-time stochastic tasks. Based on this interpretation, we can see what task difficulty is, what instance difficulty is (relative to a task) and also what task compositions and decompositions are.

Keywords: Task difficulty, task breadth, Levin’s search, universal psychometrics.

1 Introduction

There is an increased interest in artificial intelligence evaluation, motivated by recent breakthroughs produced by new technologies, and also because of an urging pressing of characterising the abilities of machines, so that we can have a better account of their implications in the job market and the potential risks. In the context of universal psychometrics [25], defined as the evaluation of cognitive features of humans, non-human animals, computers, hybrids and collectives thereof, the notion of ‘cognitive task’ was introduced and formalised, but several issues still require further development, such as the associated concepts of task difficulty and task breadth (or alternative concepts such as composition and decomposition).

In this paper, we realise that many tasks in artificial intelligence, human psychometrics and animal cognition do not fit well within the formalism of (PO)MDP, especially with the concept of ‘transition function’. With the help of some examples of cognitive tasks, we identify several features that a proper notion of cognitive task should incorporate. It is important that we realise that the evaluation setting does not need to be defined in terms of the way particular approaches solve the problem (which can still be approached through a reinforcement learning approach using a MDP formalism). What we see is that the alternate finite-state view of MDP based on transition functions makes it difficult to understand how some simple tasks, such as response time, can be accounted for, and most especially, when we want to analyse the computational complexity of the space of policies, in order to derive notions such as task difficulty.

In the case of using formalisms that rely on transition functions such as (PO)MDP (for discrete or continuous cases), the notion of computational cost must be derived from the algorithm behind the transition function, which may have a very high variability of computational steps depending on the moment: at idle moments it may do just very few operations, whereas at other iterations it may require an exponential number of operations (or even not halt). The maximum, minimum or average for all time instants show problems (such as dependency on the time resolution for which the steps of the algorithm should remain fairly constant, or the use of space with finite states). Also, the use of transition functions differs significantly

in the way animals (including humans) and many agent languages in AI work, with algorithms that can use signals and have a control of time through threads (using, e.g., “sleep” instructions where computation stops momentarily). Of course, we are not saying that it is impossible to find modifications of MDP to accommodate all this, but we are going to see a different formalism, based on probabilistic (Turing) machines with a special “sleep” instruction.

The other important thing is the notion of response, score or return R for an episode. Apart from relaxing its functional dependency with the rewards during an episode, to account with a goal-oriented task, we consider the problem of commensurability of different tasks by using a level of tolerance, and deriving the notion of acceptable policy from it. While this seems a cosmetic change, it paves the way to the notion of difficulty —as difficulty does not make sense if we do not set a threshold or tolerance— and also to the analysis of task instances.

After these instrumental accommodations, we are ready to derive the computational steps taken by an algorithm during a task. This is crucial for the notion of difficulty. With this representation, the straightforward idea of difficulty as search effort is used, whatever the kind of search is (“intellectual”, “evolutionary” or “cultural”, as Turing distinguished [57]). Difficulty is just the logarithm of the computational steps that are required to find the optimal policy, including trying several possible policies and verifying them. This is in accordance with Levin’s universal search [43, 44], the notion of information gain [17] and the interpretation of the “minimal process for creating [something] from nothing” [47]. However, we have to be very careful that when an agent interacts with the world or a task, this task can give hints and reinforce the search process. How all this is set makes a big difference, especially for the interpretation of verification (for instance, in Levin’s search, verification is simply the execution of the algorithm to check the output). It is insightful to see that in some tasks, the agent can just find policies such as ‘do what I have seen’, ‘do a Monte Carlo approach’ and ‘learn from the examples’ instead of the ‘ideal’ specific policy for the problem. These policies (or meta-policies) may require fewer computational steps during the search and may lead to acceptable policies, even if the code for the search has to be counted in the description of the policy.

The notion of difficulty for tasks is usually applied to this generation of a policy for the task, either by evaluation or through learning. This is very different to the computational complexity of the problem. For instance, one thing is to learn a function that sorts a string and another thing is to analyse whether a certain algorithm (or any algorithm whatsoever) can sort a string in a number of steps that is polynomially related to the size of the string. Of course, we can ask about the computational complexity of *learning* a sort function from examples, but in this cases we need to consider several factors such as (1) the desired sort function in terms of accepted level of error, (2) what the minimum efficiency requirement for the policy is, (3) how many examples are needed and (4) how much time is needed. Some of these questions have been solved by learning theory, and settings such as PAC learning.

In addition, the notion of task instance difficulty is more controversial, as it usually assumes that it is relative to the task (e.g., ‘30+0’ is an easy instance of the addition task) or even to the policy (e.g., ‘sort gabcdef’ is a very easy case for a particular sorting algorithm). Note that average-case complexity in complexity theory refers to how many computational steps are employed to solve a set of instances (with a distribution) given a particular algorithm —or for every possible conceivable algorithm. But one question that is not usually made is: How can we say that ‘sort gabcdef’ is easier than ‘sort gdaefcb’ without setting an algorithm or the definition of a distribution of algorithms? The key is to analyse the distribution of policies and the resources they require. Of course, this must be done relative to the task with a large (or infinite) number of instances. We will see that otherwise (if we just focus on one instance or a small set of instances), this does not make sense, as we can just rely on memorising the policy with a lookup table.

The paper is organised as follows. Section 2 starts with an example and tries to identify the features and requirements that a universal psychometric task should have to be a good evaluation task. Then it introduces a formalism, as general as possible, for this. Section 3 investigates the notion of task difficulty, and the necessary notions of effort (based on length and computational steps) and acceptability (using a tolerance level). Section 4 discusses whether the notion of task difficulty can be inherited for instances. Then we move to the notions of task composition and decomposition and their implications, and whether this allows for the definition of response curves that may be used for adaptive tests. Section 5 introduces a variant of Levin search that takes the stochasticity of tasks into account and includes a new term into Kt , which is based on the number of repetitions that are needed to verify that a policy is ϵ -acceptable with some given confidence $1 - \delta$, à la PAC (Probabilistic Approximate Correct). Section 6 closes the paper with some

comments about the related work and a few open questions and directions.

2 Tasks, trials and responses

Cognitive evaluation is performed through instruments, known as cognitive tests, which are composed of cognitive tasks. Consequently, we need to have a clear view of what a task is and how they can be compared. In [25], tasks are defined as interactive processes with asynchronous time where the final response is not *necessarily* a function of rewards. However, tasks are still based on transition functions and —partly because of this— there is no clear handling of idle times to define a proper notion of computational steps. In addition, it is unclear what happens if there is repeated testing on the same agent, and also if the agent has been gone through a previous training stage or not. Despite some extra notational burden, in this paper we will try to be explicit about all this.

2.1 Example

What do Talon the dolphin in Florida Keys [40] and Ana the sea lion in Valencia [1] have in common? Both have been tested about their ability to judge relative quantity, a task that is usually referred to as “relative numerosness”, “relative numerosity” or “relative quantity judgment”. Talon the bottlenose dolphin, for instance, was repeatedly tested with two different quantities such as the two shown in Fig. 1, and was given a reward if selected the lesser amount.

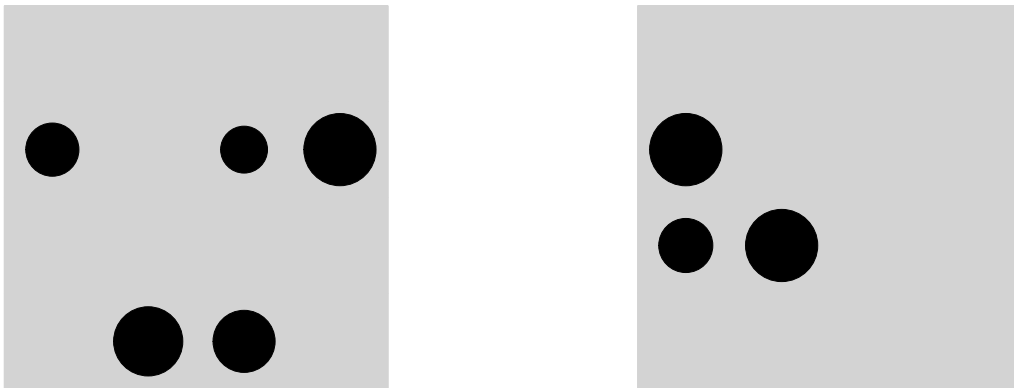


Figure 1: An example of a ‘relative numerosness’ task, where two boards are shown with a different number of dots. The size of the dots should not matter for the quantity. Left: a panel with 5 dots. Right: a panel with 3 dots.

Apart from cetaceans, many other studies about “relative numerosness” have been conducted in the area of comparative psychology, including angelfish, bears, capuchin monkeys, squirrel monkeys, cats, chimpanzees, coyotes, gorillas, hyenas, orangutans, pigeons, salamanders, sea lions and elephants (see, e.g., [1, 48, 62], to links to some of these studies).

The interesting thing about this example is, on one hand, that it has been applied to many different kinds of animals, including humans of different ages (needless to say that the task is easy for adult humans that are allowed to count). On the other hand, it is relatively easy to write a computer program that solves this task perfectly, using image recognition and simple counting. This example will serve as a running example to illustrate some issues of tasks: level of completion, stochastic character, sequentiality, training stage, etc. Also, we will use it as a good example of whether and how difficulty can be determined formally, independently of the population results.

Other tasks (originating from psychometrics, comparative psychology or artificial intelligence) will be used in what follows and will be described in more detail if needed. For instance, we will use letter series or Raven’s progressive matrices (as in IQ tests), response time, mazes, playing Pacman, English-Spanish translation, simple imitation (action equal to most recent observation), eidetic memory, sudokus and addition. These tasks are summarised in Table 1.

Id	Name	Stoc	Description	Instances and Generation
μ_{num}	Relative numerous-ness	yes	Choose between left/right panels the one with fewest number of dots	Number and size of dots uniformly chosen from a range.
μ_{RPM}	Raven's progressive matrices	yes	Choose the option that better matches the matrix	A finite set of problems, uniformly chosen.
μ_{Ctest}	C-test	yes	Find the continuation of a letter series	The difficulty of the sequence is uniformly chosen.
$\mu_{response}$	Response time	yes	Press left/right button when and as the signal indicates	A uniform distribution of delays from a range. Left/right uniform too.
μ_{maze}	Maze	yes	Go from start to exit in a maze	A random generator of solvable mazes with variable proportion of walls.
μ_{pacman}	Pacman	yes	Eat all dots without being eaten by some ghosts	Ghosts move with some patterns but stochastically.
μ_{trans}	Translation	yes	Translate a text from English to Spanish	Texts taken from a large finite corpus.
μ_{imit}	Simple imitation	yes	Repeatedly perform the action equal to most recent observation	Observation chosen uniformly from a finite set
μ_{guess}	Guess action sequence	yes	Actions are guessed until match (with reward), then another action	Sequence chosen uniformly from a finite set
$\mu_{eidetic}$	Eidetic memory	yes	Remember a sequence of numbers that have only been shortly shown	Various exposition times and sequences
μ_{srote}	Short constant string	no	The agent must output the string. Correct string is shown afterwards	Always the same text for all instances
μ_{lrote}	Long constant text	no	The agent must output the string. Correct string is shown afterwards	Always the same text for all instances
μ_{sudoku}	Sudokus	yes	A 9×9 sudoku	Consistent puzzles from a random generator
μ_{add}	Addition	yes	Addition of two natural numbers	Numbers chosen uniformly from a range.

Table 1: Some illustrative tasks that can be used to reason about some of the concepts discussed in this paper. The column ‘Stoc’ indicates whether they are stochastic or not.

2.2 Features of a task

Having a look at the ‘relative numerosness’ and other tasks, we need to consider several features (some of them present in comparative psychology, psychometrics, reinforcement learning, etc.):

- Tasks can be administered in one or more trials. There is a result or response R at the end of a task trial.
- As trials can be repeated, if the system is not reinitialised after each trial, we have a cumulative evaluation of the task. Its evolution is measured in terms of the number of trials or attempts ν .
- Asynchronous time: many tasks in psychometrics require time to be continuous or to be actual time. For instance, the response time task or a real-time Pacman requires the use of time. Note that there is a long tradition of discrete time in AI, especially in reinforcement learning, although the use of continuous time environments has also been studied in the areas of intelligent control and also in various kinds of reinforcement learning [54]. We are just in favour of using asynchronous discrete time. The crucial point is that actions and observations from the environment are not alternating.
- Trials have a limited time τ . Performance depends on this limited time.
- Interaction is given by discrete structures, but not bounded, i.e., we will consider algorithmic actions and algorithmic observations. In other words, actions and observations are complex structures that cannot be represented with a finite set of actions and observations. For instance, in the ‘relative numerosness’ or ‘Pacman’ tasks we can assume a finite grid of points up to some given resolution, but for an English-Chinese translation task inputs and outputs are, in principle, not bounded.
- States are algorithmic. There is no finite set of states. The Markov property is not assumed either. Tasks are usually non-ergodic (it is the repetition of several task instances what makes learning possible).
- Tasks (and subjects) are stochastic (if they are not stochastic—or not very stochastic—, rote learning will be frequent). Several trials for a task can give different results.
- When several instances of the *same* task are performed they can be averaged and their expected value estimated. However, it is important to note that for *different* tasks, the aggregation of the response of different tasks (e.g., an average) might not make sense (if the values are not commensurate). When using different tasks, if they are to be aggregated nonetheless, the final score of a test can depend on tolerance levels ϵ over the responses. Only if these are seen in terms of similar difficulty, the numerical aggregation (and the notion of task composition) can become meaningful.
- Rewards are a kind of transmitting supervision during a trial. They may exist or not, and may be linked or not to the response R . Difficulty will of course be affected by the (non-)existence of rewards. In any case, it is important to clarify that observations can be an indirect sign of supervision too, as we are talking about interactive tasks.
- In order to evaluate an agent, we do not need anything about the size of the algorithm behind the agent or the computational steps it requires, just whether it makes some proper actions in due time. The size of their algorithms and their computational steps are important for the calculation of the difficulty, as we will see in the following section.

The relation between repeated trials and rewards deserves some further discussion. If a task has only one trial (or the agent is reinitialised after each trial) and does not have intermediate rewards as in reinforcement learning, then the system must be necessarily predefined and specialised for that task. This is what most AI applications are conceived for. In animals, some tasks trigger an innate behaviour and can be measured in these circumstances. For instance, many animals can choose the board with the *highest* number of peanuts or fish without any training at all (and no intermediate rewards indicating whether it is doing right or not). Of course, the innate behaviour takes place because the task (or a similar one) has appeared many times in the evolutionary history of the species. However, many other tasks require some training, and this can be done in animals and in AI systems. In animals, rewards can be given at the end of an episode or during the episode. Similarly, in AI, rewards (or payoffs) are given at the end of an episode (e.g., in game theory) or during the episode (e.g., in reinforcement learning, with the reward function). Even if these two approaches exist for training, when we focus on measuring capabilities and skills, it is usual that intermediate rewards are no longer used, as their effect is more difficult to control and understand. In fact, this is not actually a

distinction between animal cognition evaluation and AI evaluation, but a distinction given by the purpose. For instance, in (video)games, it is usual that there is an intermediate reward in the form of points for a score, apart from the goals of each stage or the whole game.

From the above, it seems that for many tasks where the agent is not innate or preprogrammed for, in order to measure abilities and not specific task performance, we need to consider tasks that are both stochastic and with several trials. Several trials allow the system to be trained for the task, where the use of stochastic tasks ensures that the system does something different from rote learning (note that a large set of items chosen randomly is a stochastic task). In the case of several trials, it is important to consider that for animals (including humans) and some AI systems, reinitialisation is not possible, so we have to take into account that the realisation and result of previous trials have effect on subsequent trials. Only some tasks can avoid this effect. In fact, some tasks used in IQ tests are usually designed in such a way that there is no much interference between one exercise and the rest (in fact there is no learning or specialisation), although this effect can never be ruled out completely. Finally, in adaptive tests, dependency between trials is not only existent but characteristic. Actually, this dependency is exploited. The most general account of a task would be to consider that they are adaptive (i.e., they have memory as well), and non-adaptive tasks would be a special case. Even for a single task, we can have an adaptive test, provided we have a measure of instance difficulty or some other feature that we can use to change the distribution of instances of a task. We will deal with the issue of instance difficulty later on.

2.3 Asynchronous-time Stochastic Tasks

Now we are going to give a more formal account about how to define general tasks computationally, which comply with the features in section 2.2 above. We want interactive tasks such that, in an episode, agent and environment can exchange inputs and outputs at any time. We will first choose asynchronous time for it, as this is needed in some tasks such as ‘response time’ and other real-time problems. Apart from its need in these types of tasks, there are additional reasons for using asynchronous time in reinforcement learning [54], and artificial life [8, 12]. Even in cases where the task is alternating (e.g., a chess match), it is important to consider the time for each turn and the thinking time (one can think while the opponent is thinking, and both thinking times have to be considered).

Synchronous (or more precisely, alternating discrete time) interactive machines are based on a transition function, which is applied at each time point to change the state. The most common example is (PO)MDP. The transition function takes a state, and observation and a reward and produces an action. It goes from state to state indefinitely (even if it remains in the same state forever, there is some computation to apply at each time moment, the transition function). Transition functions can have access to the environments’s memory. In this case, if the memory is not bounded we have an infinite number of states (no longer an MDP). In any case, even with a finite number of states because of the stochastic character there might be a different number of computational steps taken for each transition (there might even be some transitions that do not halt).

Asynchronous environments are not continuous-time POMDPs, which are based on transition functions and are handled with differential equations. In fact, synchronous environments are a special case of asynchronous environments where the environment waits for the agent’s action to issue observations and rewards. Intermediate rewards during the episode are also considered but, unlike synchronous environments, the correspondence of the total result as a discounted sum of rewards is not possible in general. In fact, the number of rewards per unit of time is not limited, so the final function that maps rewards to a result may be very varied (and it is part of the definition of the task). This is similar to the way rewards were defined in [25], an internal thing given to the agent, whereas the score or response for the episode was an external thing not necessarily given to the agent.

Let us now give the definition of asynchronous-time interactive systems. In an asynchronous-time interactive system, there is a common time (which can be discrete or continuous, and can be virtual or real). Time will be shared by all systems that interact. An interactive system is a machine with a program code, a finite internal discrete memory, one or more finite read-only discrete input (tape) and one or more finite write-only discrete outputs (tape). Agents and environments use the above definition and are asynchronous-time interactive systems. The inputs of agents are called observations and the outputs are called actions. As special features, these machines have access to a read-only time measurement and a source of randomness

(either by an additional random instruction or a random tape). The programs for tasks and agents are constructed with a set of instructions that, if memory were infinite, would make the machine Turing-complete, and ultimately equivalent to a Turing machine, denoted by its program over a reference universal prefix Turing machine U . This makes this definition very close to probabilistic Turing machines¹ —which are not exactly the same as non-deterministic Turing machines. In a probabilistic Turing machine, only one course of action is taken, and no parallel computation is performed to keep all the alternative courses of actions. In fact, computable stochastic processes are usually associated to probabilistic Turing machines, and not to non-deterministic Turing machines.

We have already said that the machine will have access to a random source (through an instruction or an extra random tape). Some animals (e.g., flies, preys) behave in a random way to avoid being predated, and this behaviour does not require a very complex mechanism, just a few neurons being triggered on some environmental magnitudes acting as random number generators. There is also access to time, which can be physical time, an approximation or a virtual time. But most importantly, for the purpose of the analysis of computational steps, we consider that the machine will be able to stop momentarily, until a given time, through an instruction or special state **sleep**(t), which sets the machine to sleep until time t . During the time the machine is sleeping, no operation is performed.

Some tasks will also have intermediate rewards. Rewards are just given through another extra tape, and are interpreted as a natural number. Rewards are optional. In case they exist, the result of an episode may depend on the rewards or not. This is important, as the general use of rewards in reinforcement learning, especially with discounted reward or through averaging gives the impression that the final result or response of an episode must always be an aggregation. For instance, in a maze, an agent may go directly to the exit and may require no reward. On the contrary, a more sluggish agent may require more positive indications and even with them cannot find the exit. Rewards can be just given to help in the finding of the solution, which does not mean that the higher the rewards the higher the results. Finally, the agent is able to see the result or score of an episode (a rational number) at the end through another special tape. A final reward can be given instead of or jointly with the result.

While this is certainly more complex than other models of interactive machines, it accommodates the intuitive notion of task in many natural and artificial scenarios.

2.4 Trials and results

We consider tests that are composed of tasks (also called environments), usually denoted by μ , and are performed by agents (also called policies or subjects), usually denoted by π .

The expected value of the response, return or result of π in μ for a time limit τ is denoted by $\mathbb{R}^{[\tau]}(\pi, \mu)$. The value of τ will be usually omitted as it is understood that it is part of the description of the task μ . The \mathbb{R} function always gives values between 0 and 1 and we assume it is always defined. If the agent goes into a non-halting loop and stops reacting, this is not perceivable externally and may even lead to some non-zero \mathbb{R} (from the previous actions or because of the type of task).

Now we need to extend the notation of $\mathbb{R}(\pi, \mu)$ to consider several instances of the same task. Each attempt of a subject on one of the task instances is a trial or episode. $\mathbb{R}^{[\rightarrow \nu]}(\pi, \mu)$ returns the expected response of μ per trial with ν consecutive episodes or trials by the same agent π *without reinitialisation*². So actually it is not the same π each time, if the agent has memory. ν refers to the evaluation trials, which are used for the expected response (which is an average of all the evaluation trials and not a sum). Note that the expected response is given because π is non-deterministic and may lead to different situations from the very beginning. The distribution of what each instance of a trial will look like is inside the stochastic task. According to the task, the same instance can appear more than once, as in a sample with replacement. As the task can have memory, we can also have some tasks that are really working as if a no-replacement sampling were taking place. In order to do that, the task itself must keep track of the instances that have

¹Probabilistic Turing machines with finite tapes (except the random tape) are like “probabilistic linear bounded automata”. This is exactly the type of computers we are used to and the ones we are able to build with the current paradigm. Note that this is different to subrecursive programming systems and other models of computation where it can be determined whether programs terminate, i.e., and even what they will compute.

²Note that, if the test is not adaptive, instances have no memory, as they start from scratch. This ‘stochastic repeatability’ is related to some other conditions (e.g., ergodicity) that are sometimes imposed or assumed on tasks where a pattern or some properties can endure indefinitely.

appeared or must use some kind of randomised enumeration. Also, tasks can be adaptive. In other words, instead of talking about sampling with or without replacement, it is the definition of the task that defines this.

No waiting time or stop is considered between trials. If a task requires some resting time between trials, then this has to be included in the very trial and not in between trials.

With $\mathbb{R}^{[\rightarrow 1]}(\pi, \mu)$, or simply $\mathbb{R}(\pi, \mu)$ we denote that there is only one episode or trial (no repetitions). For instance, many tests are of this kind if items are completely unrelated, so each item has no influence on the following ones, although it is more applicable when we consider that the agent has no memory (or is reinitialised between trials). In general, especially if the items are related, for every $\nu > 1$, we have that $\mathbb{R}^{[\rightarrow \nu]}(\pi, \mu) \neq \mathbb{R}^{[\rightarrow 1]}(\pi, \mu)$ unless the agent has no memory between episodes.

Tasks with high values of τ will imply that episodes are long, while high values of ν mean that we make many repetitions. Note that some abilities are related to good results after very few repetitions (i.e., to understand a concept fast). This speed is understood in many ways, but one is clearly how many ‘examples’ or ‘instances’ are needed. Note that many machine learning techniques require many examples (e.g., deep learning [3]), many repetitions (e.g., Q-learning [63]) or large τ (e.g., AIXI [60]). In the previous example about the ‘relative numerosness’, τ is not very relevant as the task displays the boards (or panels or dishes) for a few seconds (τ may be 5 physical seconds). However, ν is important, and we usually require a number of training trials (so that the animal can learn the task) and then a series of test trials.

Note that if the *task* has no memory, this does not allow for an evolving distribution (e.g., a kind of task first and then switch to other tasks, or some kind of cumulative or adaptive tasks). Tasks with memory would be useful for adaptive tests. In this paper, unless stated otherwise, all constructs are valid with tasks with memory, even if we do not explore adaptive tests, just the fundamentals (such as difficulty, which is required for adaptive tests).

2.5 Examples

In Fig. 1, we saw an example of the ‘relative numerosness’ task μ_{num} . This can be seen as a stochastic task class where the agent sees two rectangular grids (representing plates) where we have some black spots on it. The action is just choosing left or right. If the choice is correct, the agent receives a response (and reward) of 1. Otherwise, it receives 0.

For this task (μ_{num} in Table 1), we have 4×4 ‘cells’, with the number of dots in each panel going uniformly from 1 to 16. The size of each dot is uniformly distributed between 0.2 and 1, with 1 being the diameter of the cell. In case the two panels had exactly the same number of dots, the pair would be discarded and a new one would be generated. The use of different dot sizes is used to prevent subjects from choosing the panels exclusively (or mostly) by their overall darkness (if there are more dots and all are equal sized then the panel is always darker overall). In many studies, 80% success rate is considered as a level where the subject is considered to perform the task successfully.

It is relatively easy to implement an agent that processes the image, recognises the shapes and counts the dots. However, we are interested in seeing that it is also possible to score well in this task with an agent that does not count at all. This agent, π_1 performs a Monte Carlo approach and (virtually) throws n points randomly inside the panel. It calculates the darkness of the panel as the percentage of points that are black (i.e., it is inside a dot). At the end, the darkness of both panels is compared and the least dark is chosen. If π_1 uses $n = 100$ points for each panel, the agent is able to score 0.8675. Note that even if there are $(4^2 - 1) \times (4^2 - 2) = 210$ different number comparisons, the possible cell locations of the dots and their different sizes make a virtually infinite number of different instances. Different results are obtained if the number of points of the Monte Carlo method is changed. For instance, if π_2 only uses $n = 50$ then $\mathbb{R}(\pi_2, \mu_{num}) = 0.8495$. Still, if π_3 only uses $n = 10$ then $\mathbb{R}(\pi_3, \mu_{num}) = 0.746$. Actually, with just one point, π_4 can still do significantly better than random: $\mathbb{R}(\pi_4, \mu_{num}) = 0.575$. Clearly, the computational cost decreases from π_1 to π_2 .

The response time task ($\mu_{response}$ in Table 1) is an interesting task to analyse. We could have a policy π_1 that is constantly checking the input to see if a response is needed. Assuming very high speed (e.g., it can check the input, process it and see whether it has to react or not one million times per second), this π_1 would score almost perfectly. However, it would also use many computational steps. Another algorithm π_2 could just check 10 times per second (by using the instruction `sleep(t)`, with $t = 0.1s$), and get a reasonable

good result with much less computational cost than π_1 (it is not exactly 100,000 less because when the signal is not there the instructions to be executed are expected to be fewer than when the signal is there).

These two examples stress the issue of computational complexity and how it is interpreted in asynchronous tasks.

3 Task difficulty

The first thing to clarify about difficulty is whether we apply it to the *generation* of the policy or the *application* of the policy. The generation phase can be innate (by programming or nature) or acquired (through training or learning). In comparative psychology and artificial intelligence it is usual to have these two phases. It is very important to determine which phase we are referring to when talking about difficulty. For instance, if we evaluate the ability of an animal of being able to do a task that involves counting, what we want to know is whether the animal can acquire this ability. If we evaluate the ability of making calculations (e.g., addition), we are clearly assuming that the system already has the algorithm for addition, and we are just examining how well they do. This is clearly the case in many specific-task evaluation, such as driving a car, game-playing, etc. The confusion comes because in both cases we will evaluate the performance on the task in the same way.

Despite the same evaluation, the notion of difficulty must be understood very differently. For instance, the difficulty of the generation phase usually refers to tasks with many instances (how difficult is it to learn to add from examples), while application usually refers to instances (how difficult “3+2” is compared to “234+998”). In this section we will focus on task difficulty, leaving instance difficulty for the next section. For instance, in the relative numerosness task, the generation difficulty depends on how much it takes to program the algorithm for this task, the evolutionary cost of acquiring the algorithm or the learning cost of acquiring the algorithm.

The difficulty of solving a stochastic task can be assessed by [45] (1) looking at the complexity of the task (this is known as a structuralist approach), (2) looking at the complexity of the policy (or the resources that are required by the subject) or (3) looking at the interaction between task and subject.

Our view of the generation difficulty is an “algorithmic difficulty”, which is basically the computational steps required to build the policy algorithm, which depends on the tolerance level of the task, the interaction and hints given by the task, the algorithm length, its computation cost and its verification cost. We now see all these components below.

3.1 Agent resources, acceptability and interaction in asynchronous environments

The first thing we will require is the length of a policy. The length of an object x , denoted by $L(x)$ expresses the length of a string using a binary code for the object. This function can be applied to tasks and agents. There is an important thing to consider here. If a program has the ability to self-modify, as it happens with self-improvement agents, then when we measure $L(\pi)$ of an agent π during a series of trials, the value might change. However, one program can get extremely short by moving all the code to memory. Consequently, analysing the evolution of the program during the execution of several trials is like analysing how memory is evolving, so we will just consider the program π as it was before the evaluation.

The second thing we will require is the computation steps taken by a policy. In synchronous environments, one option may be to add all the steps taken for all time cycles, but this clearly depends on the resolution of the discrete time. Also, many transition functions may be just idle transitions, where the agent is just checking whether something is happening. But imagine an agent that wants to wait for 10,000 transitions. Even if very few operations are executed in each transition, these transitions count. To avoid this problem another option is to calculate the maximum, as done in [22] with the so-called Kt^{max} . This is a very rough approximation, as one single peak can make this very large. The mean or sum do not behave better, either.

Fortunately, here tasks are defined as asynchronous. When the agent needs to wait until a situation or time is met, if the instruction `sleep(t)` is used, we should not consider all these ‘waiting’ times for the computational steps. With this interpretation, the expected³ execution steps of π per trial when performing

³This has to be ‘expected’ if we consider stochastic environments or agents.

task μ are denoted by $\mathbb{S}^{[\mapsto \nu]}(\pi, \mu)$ for a time limit (τ) given by the task for each trial. Note that we consider all the computational steps performed by π during all the ν *evaluation* trials for this expected value (they are not added, though). If at any moment π enters an infinite loop⁴, then $\mathbb{S}^{[\mapsto \nu]}(\pi, \mu)$ is infinite. As we are using stochastic agents and environments, it is sufficient that one possible combination of the trials leads to non-termination such that the expectation is infinite.

The third thing is about memory requirements (space). What if a policy requires much more memory than another? This is also important in the context of several trials if the policy requires the memorisation of the information of previous trials. We would do similiary for the internal memory used by the policy, considering that there are instructions to ask for more memory and free memory (or we can record up to where the algorithm reaches if there is an internal tape). The notation is $\mathbb{M}^{[\mapsto \nu]}(\pi, \mu)$. In this paper we will not consider space because (1) the use of n bits of memory requires at least n computational steps, so the latter are going to be considered anyway and (2) steps and bits are different units.

The fourth thing is verification. When we discuss the effort about finding a good policy, there must be some degree of certainty that the policy is reasonably good. As tasks and agents are stochastic, this verification is more cumbersome than in a non-stochastic case. We will discuss about this later on in the paper.

For the moment, we will just combine the length of the policy and the computational steps, by defining $\mathbb{LS}^{[\mapsto \nu]}(\pi, \mu) \triangleq L(\pi) + \log \mathbb{S}^{[\mapsto \nu]}(\pi, \mu)$. Logarithms are always binary. We will explain later on why we apply a logarithm over \mathbb{S} .

The fifth thing is the tolerance level of the task. In many cases, we cannot talk about difficulty if there is no threshold or limit for which we consider a policy acceptable. For instance, how difficult is a response time task? It depends on where we put the threshold. How difficult is pacman? It depends on how many points or time we want to achieve. It is true that some tasks have a response function R that can only be 0 or 1, and difficulty is just defined in terms of this goal. But many other tasks are not binary (goal-oriented), and we need to establish a threshold for them. In our case, as the return function R goes from 0 to 1, we can take 1 as the best response and set the threshold on $1 - \epsilon$. With this we first consider the notion of acceptability.

We define acceptability in a straightforward way. The set of acceptable policies for task μ given a tolerance ϵ is given by

$$\mathcal{A}^{[\epsilon, \mapsto \nu]}(\mu) \triangleq \{\pi : \mathbb{R}^{[\mapsto \nu]}(\pi, \mu) \geq 1 - \epsilon\} \quad (1)$$

Note that the combination of the expected value with a tolerance greater than 0 makes that the agent can do terribly wrong in a few instances, provided it does well on many others. While the expected value corresponds to the mean, we could use another statistic.

The sixth thing is the interaction and hints given by the task. This can be during the task (through rewards or other observations) or throughout several trials. During the task, algorithms can use past experience and rewards to solve the task. For instance, μ_{imit} in Table 1 can be solved by simply observing and copying, so actually the policy is an algorithm that does this. Similarly, if we have an agent that is not reset after each trial, the algorithm can just learn from previous trials. For instance, μ_{rote} in Table 1 is solvable by an algorithm that memorises the correct string from a previous trial. In general, we can have many different kinds of policies: ‘**forever do action 1, wait(1), do action 2, wait(1)**’, which ignores the observations from the task completely, ‘**forever output what μ outputs**’, which uses observations but ignores previous trials, ‘**execute code1, if result of previous trial is lower than 0.5 then execute code2 in the following trials**’, which uses the results of previous trials, ‘**execute random actions every 1 units of time. Memorise those actions that generate some change of observations. Repeat them on the following trials**’, which uses the observations of previous trials, and ‘**execute random actions every 1 units of time. Memorise those actions that receive positive rewards. Repeat them on the following trials**’, which uses the observations and rewards (if there are) of previous trials. But some other ‘meta-algorithms’ are equally valid, such as ‘**try algorithms randomly from a given set of algorithms. If one has been good for the past five trials, use it for ever**’ or ‘**use search heuristics of type 1 for the first 100 trials. If unsuccessful,**

⁴Here, we are not concerned about halting, but rather that the number of steps is finite before the time limit τ .

switch to heuristics of type 2'. These are just examples of whatever algorithm that can be used, including self-improving algorithms.

The consideration of stochastic tasks is fundamental. For instance, consider the “relative numerousness” task μ_{num} again in Table 1. For each instance, the solution is just ‘left’ or ‘right’. The information that is needed is just one bit. If we just put one possible instance in a task (i.e., a non-stochastic task) —and we knew that the task is not stochastic—, then just one repetition of the *same task instance* would be enough to find the solution, which will be very short (e.g., ‘**choose the left board**’). If the task can generate a great or infinite number of instances, then the possibility of rote learning is reduced, and the policy would incorporate some generalisation.

3.2 Difficulty as minimum resources

Having the the above issues into account we can define a first parametrised version of difficulty. Bear in mind that these are general expressions whose goal is to understand what a function of difficulty is. In many tasks, though, we may use a more practical (and particular) function of difficulty.

And now we are ready to link difficulty to resources. This is usual in algorithmic information theory, but here we need to calculate the complexities of the policies (the agents) and not the problems (the tasks). So, our first approach is to evaluate difficulty as the length of the shortest acceptable policy:

$$K^{[\epsilon, \mapsto \nu]}(\mu) \triangleq \min_{\pi \in \mathcal{A}^{[\epsilon, \mapsto \nu]}(\mu)} L(\pi) \quad (2)$$

The use of the notation K and the structure of the definition make it clear that this can be understood as a version of Kolmogorov complexity for tasks, where instead of talking of the shortest program that generates a string, we talk about the shortest program that solves the task.

Note that K is not only parametrised with a tolerance ϵ but also with the number of evaluation trials. So our notion of difficulty depends on these parameters. We could think about letting be unlimited, so we would have $K^{[\epsilon, \mapsto \infty]}(\mu)$. This allows programs that use several trials, so we can have a policy π that just does ‘**enumerate all possible programs and execute each of them on as many trials as needed and choose the best one for the subsequent trials**’. Let us call this strategy⁵, $\pi_{find-L-best}$. Assuming there is a finite acceptable policy, the length of this program $\pi_{find-L-best}$ could be taken as an upper bound for K because this program is going to find the policy if given infinite trials, just by enumeration. For some tasks, of course, there might be other programs that could be shorter than $\pi_{find-L-best}$. For instance, in the simple imitation task π_{imit} , it is expected that the coding of the program ‘**copy the observation to the action**’ is shorter than $\pi_{find-L-best}$. Examples for some of tasks are shown on Table 2.

We can also consider $K^{[\epsilon, \infty, \mapsto 1]}(\mu)$, but in this case it cannot be a program that searches for the policy across several trials. For some kinds of tasks, especially those that do not give partial indications during the task, this will account for the shortest policy that gives an ϵ -acceptable solution *without looking at the task at all*. For others, the task will provide the required information (like an input) but the interaction will be just that. For instance, in the relative numerousness task, depending on the tolerance, the Monte Carlo policy could be a good option, as it is a very short policy. Actually, a version of the Monte Carlo with a huge amount of points would be better, disregarding its high computational cost, since computational steps are not taken into account. For the simple imitation task ‘**copy the observation to the action**’ would still be chosen. It may seem counterintuitive to analyse a situation with just one trial with a policy that cannot be found with just one trial (the chances are actually about 2^{-L}), but here we are trying to measure difficulty. Examples for some of the tasks are shown on Table 2.

The problem about K is that it does not take computational cost into account (this also makes it uncomputable). A common solution, inspired by Levin’s Kt (see, e.g., [43] or [44]), is to define:

$$Kt^{[\epsilon, \mapsto \nu]}(\mu) \triangleq \min_{\pi \in \mathcal{A}^{[\epsilon, \mapsto \nu]}(\mu)} \mathbb{LS}^{[\mapsto \nu]}(\pi, \mu) \quad (3)$$

Note that the above has two expectations: one in \mathbb{LS} and another one inside \mathcal{A} . The interpretation of the above expression is a measure of effort, as used with the concept of computational information gain with Kt

⁵In a way, this strategy is like an AIXI-like algorithm [34].

Task	$K^{[\epsilon, \mapsto \infty]}(\mu)$	$K^{[\epsilon, \mapsto 1]}(\mu)$	$Kt^{[\epsilon, \mapsto \infty]}(\mu)$	$Kt^{[\epsilon, \mapsto 1]}(\mu)$	I
μ_{num}	$\pi_{find-L-best} \Rightarrow$	‘Monte Carlo policy with many points’	$\pi_{find-LS-best} \Rightarrow$	‘Monte Carlo policy with a few points’	M
μ_{RPM}	$\pi_{find-L-best} \Rightarrow$	‘Shortest rpm solver’	$\pi_{find-LS-best} \Rightarrow$	‘LS-optimal rpm solver’	M
μ_{Ctest}	‘Monte Carlo search on sequence patterns’	‘Monte Carlo search on sequence patterns’	‘Levin search on sequence patterns’	‘Levin search on sequence patterns’	-
$\mu_{response}$	‘react with minimum sleep periods’	‘react with minimum sleep periods’	‘react with fair sleep periods’	‘react with fair sleep periods’	-
μ_{maze}	‘right-hand traversal’	‘right-hand traversal’	‘LS-optimal traversal’	‘LS-optimal traversal’	-
μ_{pacman}	$\pi_{find-L-best} \Rightarrow$	‘eat and escape from predators’	$\pi_{find-LSbest} \Rightarrow$	‘eat and escape from predators’	M
μ_{trans}	$\pi_{find-L-best} \Rightarrow$	‘shortest-translator’	$\pi_{find-LSbest} \Rightarrow$	‘LS-optimal-translator’	M
μ_{imit}	‘copy the observation to the action’	‘copy the observation to the action’	‘copy the observation to the action’	‘copy the observation to the action’	-
μ_{guess}	‘guess randomly until reward’	‘guess randomly until reward’	‘guess randomly until reward’	‘guess randomly until reward’	-
$\mu_{eidetic}$	‘repeat what has been seen’	‘repeat what has been seen’	‘repeat what has been seen’	‘repeat what has been seen’	-
μ_{srote}	‘output decompressible TEXT’	‘output decompressible TEXT’	‘efficiently decompressible TEXT’	‘efficiently decompressible TEXT’	-
μ_{lrote}	‘copy text from previous trial’	‘output decompressible TEXT’	‘copy text from previous trial’	‘efficiently decompressible TEXT’	H
μ_{add}	‘addition by incrementing’	‘addition by incrementing’	‘efficient addition’	‘efficient addition’	-
μ_{sudoku}	‘exhaustive sudoku search’	‘exhaustive sudoku search’	‘efficient sudoku solver’	‘efficient sudoku solver’	-

Table 2: Some of the illustrative tasks defined in Table 1 and the kind of policies that could lead to the minimisation of the complexity measures K or Kt with or without history. The cases with $\mapsto 1$ are blind to previous trials, either because there is not any previous trial or because the agent has no memory or is reinitialised for each trial. For those where $\pi_{find-L-best}$ or $\pi_{find-LS-best}$ appears, we assume there is no better policy (in terms of L or LS) that achieves ϵ . The last column shows the few cases where there is a difference between many trials or just one trial. This effect of incrementality can be reflected in terms of algorithm self-improvement or meta-search, represented by M (and we also show a right arrow meaning that in the end it will be executing the algorithm on the right), and the use of history in other ways, H.

in [17].

We first consider $Kt^{[\epsilon, \mapsto \infty]}(\mu)$. With this we allow for as many trials during evaluation. In other words, effort can be put in finding the policy, but the policy must be efficient. Again, this would sometimes end up choosing ‘enumerate all possible programs and execute each of them on as many trials as needed and choose the best efficient one’. This happens because despite its enormous computational cost, when the trials go to ∞ the algorithm may finally find the particular policy and start exploiting it. As it is the expected value for the infinite number of trials that counts, this policy is efficient for an infinite number of trials. Let us call this strategy⁶, $\pi_{find-LS-best}$, which is again of not much practical use. Examples for some of the task are shown on Table 2.

We can compare with $Kt^{[\epsilon, \mapsto 1]}(\mu)$. In this case, the meta-policies such as $\pi_{find-LS-best}$ are avoided, but we have that the policy cannot take advantage of previous trials. In a way, this version is measuring difficulty when the agents have no memory (or are reinitialised).

All these options are summarised in Table 2, which shows the tasks introduced in Table 1 with the values of several difficulty functions. We see that in some cases, the previous trials or part of the trial itself can be used to learn a pattern (as shown in the last columns).

The cases of μ_{srote} and μ_{lrote} are significant. Both are just non-stochastic tasks that can be just done by rote-learning once a couple of instances are seen. In fact, this is an extreme case of stochastic tasks where there is a relevant part that is constant. We see that both are considered simple when there are several trials (either by memorising a short string or by using a policy that just memorises and copies it from the previous instance). The use of this copy&paste policy can only be appreciated when the size of the thing to be copied

⁶In a way, this strategy is like an AIXItl-like algorithm [60].

has a certain value (for short strings, nothing can beat the policy with the string itself). This is a crucial example of why a blind search that tries to find policies without looking (and learning from) previous trials can be less efficient than another looking at previous trials. In other words, in an interactive scenario, an enumeration-like search might not be the best thing to do. This has been realised in some modifications of Levin’s universal search for agents.

Kt puts together the length of the policy and the computational steps it takes, including both searching and execution. This makes it consider any meta-search procedure inside the policy, provided that this (with the information of the task) is more effective than getting the policy from nothing. In other words, if the task gives hints and there is a short and fast procedure that can use these hints to find the policy, then the expression of Kt will give this policy. Anyway, the use of Kt is related to Levin’s universal search [43, 44], as if we measure the computational steps that are required to find the algorithm that minimises Kt we have to go approximately through $2^{L(\pi)}$ programs with their corresponding execution steps of S . By multiplying these two terms and calculating a binary logarithm we have Kt . This connection, which will be better described later on, allows us to define the unit of difficulty as the logarithm of computational steps.

We have said why $Kt^{[\epsilon, \rightarrow \infty]}(\mu)$ is not completely satisfying, since for some problems, the meta-search policy $\pi_{find-LS-best}$ is chosen. The reason is that, despite the great computational effort of $\pi_{find-LS-best}$, this can concentrate on the first millions of trials and then progressively switching the behaviour so that the best policy so far is used. The problem is because we are calculating Kt for an infinite number of trials. We have also discussed that $Kt^{[\epsilon, \rightarrow 1]}(\mu)$ cannot take history of previous trials.

An option as an upper-bound measure of difficulty in between would be $h(\mu) \triangleq Kt^{[\epsilon, \rightarrow \nu]}(\mu)$, for a finite ν and given ϵ . That means that any search has to be done during evaluation and the computational steps here will be taken into account (if ν is not too large). In general, if ν is very large, then the last evaluations will prevail and any initial effort to find the policies and start applying them will not have enough weight. On the contrary, if ν is small, then those policies that invest in analysing the environment will be penalised. That means that we will need to invest as little computation steps and trials to find an acceptable policy and then execute it for as many trials as needed to make $\mathbb{R} \geq 1 - \epsilon$. This is in a way a trade-off between exploration and exploitation. It also requires a good assessment of the metasearch procedure to *verify* the policy so it can go to exploitation. In any case, the notion of difficulty depends, in some tasks, on ν . We will come back to this issue later on, as we will analyse the ‘verification cost’, and how the number of trials ν can be derived by a confidence degree such that the policy solving the problem is found and the trials can stop.

4 Task instances, task composition and decomposition

Up to this point we have dealt with a first approach to *task* difficulty. A task includes (infinitely) many task instances. What about *instance* difficulty? Does it make sense? In case it does, instance difficulty would be very useful for adaptive tests, as we could make the stochastic task adaptive and start with simple instances and adapt their difficulty to the ability of the subject (as in adaptive testing in psychometrics).

However, there are many confounding factors to determine the difficulty of a single instance. For instance, for a division task we may have these two instances: $6/3$ and $1252/626$. If the task is stochastic and includes many divisions, a policy that actually makes divisions will pay off. But if we create a task with just $6/3$ or $1252/626$ as only instances, in both cases the solution would be just 2, which is not only equal for both instances, but also a value that has no relation whatsoever to the difficulty of these instances.

The key issue is that instance difficulty must be defined *relative to a task*. At first sight, the difference in difficulty between $6/3$ and $1252/626$ is just a question of computational steps, as the latter usually requires more computational steps if a general division algorithm is used. But what about $13528/13528$? It looks an easy instance. Using a general division algorithm, it may be the case that it takes more computational steps than $1522/626$. If we see it easy is because there are some shortcuts in *our* algorithm to make divisions. These shortcuts are frequently applied instead of the general procedure. One of the shortcuts would be to return 1 if both arguments are equal. Of course, we can think about algorithms with many shortcuts, but then the notion of difficulty depends on how many shortcuts it has. In the end, this would make instance difficulty depend on a given algorithm for the task (and not the task itself). This would boil down to the steps taken by the algorithm, as in computational complexity. For the relative numerosness task, for instance,

the difficulty of an instance would be radically different if we are thinking about a counting policy (for which all instances are approximately equally easy) or we are thinking about a Monte Carlo policy (which depends on the difference in the total area of the circles, as the algorithm can stop when the difference is statistically significant).

We can of course take a structuralist approach, by linking the difficulty of an instance to a series of characteristics of the instance, such as its size, the similarities of their ingredients, etc. This is one of the usual approaches in psychology and many other areas, including evolutionary computation, but does not lead to a general view of what instance difficulty really is. For the divisions above, one can argue that 13528/13528 is more regular than 1252/626, and that is why the first is easier than the second. However, this is false in general, as 13528^{13528} is by no means easier than any other exponentiation.

Some other approaches also link the difficulty of an instance or problem to the “probability of failure” [7] or to the “probability-of-failure and mean time-to-solution” [4]. The probability of failure can be defined in terms of one policy (so we would have again a notion of difficulty dependent to the best policy solving the task), but another perspective is “the likelihood that a randomly chosen program will fail for any given input value” [7]. This is interesting. Apparently, it looks like the population-based approach in psychology (apply the instance to some individuals and record times and success rates), as it is based on a population of programs.

Here, we have several problems to follow this idea. We would need a population⁷. Also, we have that difficulty depends on computational cost and success rates, which are expressed in very different units. If the difficulty of a task is 8 (in logarithm of steps), what does it mean if we say that one of its instances has a difficulty of 0.3 (in proportion)? In any case, we may agree that computational cost and success rate are relevant, but they do not work in this way as a function of difficulty.

4.1 Instance difficulty as rareness

Instead of considering all policies⁸, we can consider the best policy. The insight comes when we see that best policies may change with variable values of ϵ . This leads to the view of the relative difficulty of an instance with respect to a task *as the minimum LS for any possible tolerance of a policy such that the instance is accepted*.

In order to formalise this concept, we must first formalise the notion of instance. For stochastic tasks, an instance is simply the very task for which its random behaviour is fixed. This can be obtained with the underlying model by setting a fixed string to the random tape or by setting a seed to the random generator (as in many computer languages). We denote by μ^σ an instance of μ by setting seed σ .

We first define the set of all optimal policies for varying tolerances ϵ_0 as:

$$Opt_{\mathbb{LS}}^{[\mapsto \nu]}(\mu) \triangleq \left\{ \arg \min_{\pi \in \mathcal{A}^{[\epsilon_0, \mapsto \nu]}(\mu)} \mathbb{LS}^{[\mapsto \nu]}(\pi, \mu) \right\}_{\epsilon_0 \in [0,1]} \quad (4)$$

And now we define the instance difficulty of μ^σ with respect to μ as:

$$h^{[\epsilon, \mapsto \nu]}(\mu^\sigma | \mu) \triangleq \min_{\pi \in Opt_{\mathbb{LS}}^{[\mapsto \nu]}(\mu) \cap \mathcal{A}^{[\epsilon, \mapsto \nu]}(\mu^\sigma)} \mathbb{LS}^{[\mapsto \nu]}(\pi, \mu) \quad (5)$$

The interpretation of the formulae above is as follows. Take all the optimal policies (in terms of LS) for varying values of ϵ . Sort them by their ϵ increasingly. The first one that is acceptable for μ^σ gives *the best policy for μ that covers μ^σ* . The LS of this policy is the relative difficulty of μ^σ with respect to μ . Note how the order of the minimisation is arranged in equations 4 and 5 such that for the many policies that only cover μ^σ but do not solve many of the other instances, these are not considered because they are not in $Opt_{\mathbb{LS}}$.

Let us see this for the relative numerosness task. Imagine the instance in Figure 1. Let us choose a task tolerance $\epsilon = 0.1$, which we call the reference tolerance. Now consider all the possible policies solving

⁷We could assume a universal distribution of policies. This is related to the solution presented in this paper, since the shortest policies have a great part of the mass of this distribution.

⁸As said above, we could also consider a universal distribution of policies, which would give a high probability to the best policy.

the original task (considering many instances) when we vary the tolerance. For instance, for tolerances ϵ_0 from 1 to 0.5 we have that the best policy is most likely one that always chooses left (or right), assuming that we have a balanced proportion of instances where the answer is left or right. For these tolerances, the error of this policy will always be acceptable. However, the error for μ^σ will be worse than the reference task tolerance level sets ($\epsilon = 0.1$). The interesting thing comes next, when we increase the tolerances. There might be a policy for tolerances ϵ_0 0.3 or 0.2 such that is also a policy for μ^σ with the reference task tolerance $\epsilon = 0.1$. In this case, the \mathbb{LS} of this policy would be the difficulty of the instance. In other words, difficulty of an instance is the minimum effort for the whole task such that the instance is well covered.

This notion of relative difficulty is basically a notion of consilience with the task. If we have an instance whose optimal policy is unrelated to the optimal policy for the rest, then this instance will not be covered until the tolerance becomes very low. Of course, this will depend on whether the algorithmic content of solving the instance can be accommodated into the general policy. This is closely related to concepts such as consilience, coherence and intensionality [16, 26, 30, 18, 17]. If the instance is an outlier then it will be more difficult because it requires extra information to accommodate into the policy but also because the probability that it appears as part of the policy from the task is very low and hence it is hard that this case could be covered. In a way, difficulty is a notion of ‘rareness’ —in some senses of the term, special and unlikely.

A different case is when there are many instances of some kind that are different from the rest of instances in the task. In this case, it is not an instance that is rare, but a set of instances, and it is better to analyse this in terms of task decomposition, as we will see in the following section.

Finally, it must be said that equation 5 might be undefined for some instances, as none of the optimal policies for varying values of ϵ_0 is able to cover it appropriately. This of course implies that in these cases there is no policy for the task with no tolerance ($\epsilon_0 = 0$). This is related to whether we define tolerance with respect to 1 or with respect to the best policy. In the latter case, the acceptable policy with no tolerance would always exist. But still, some instances might not be covered. That does not imply necessarily that there are no policies for these instances, but that there is no acceptable policy for these instances such that it is also an acceptable policy for all the other instances.

We can now see another example. For instance, in a task where the agent has to guess the following symbol in a letter series, such as the task μ_{Ctest} in Table 1 or Thurstone’s letter series [56], we may wonder why the series *aaaaaaaa* seems easier than *aacaeag*. Two explanations are here. First, given the previous definition, we can see that for high tolerance levels some simple policies may solve some series (e.g., a program that just solves arithmetic and geometric series would solve the first but not the last one). As a result, these simple incomplete policy would score some results if arithmetic and geometric series are a relevant proportion of all series. This is exactly what the program passing IQ tests from [50] did, using some predefined rules for some common sequences. This would actually give a grading of instances, which some of them being in the same class (each class given by each of the policies returned by 4). Second, we can of course assume the best policy overall (e.g., by considering the given tolerance ϵ or tolerance 0). The policy in this case, as shown in Table 2, would be a kind of Levin’s search on the possible patterns. The difficulty would just be the computational steps of using this algorithm for the policy. As we mentioned above, there is a connection between the logarithm of the steps required by Levin’s search and Kt , the measure of instance difficulty that was used in the C -test [29, 15].

Both explanations are sufficiently compelling to see whether both can be combined. A mixture of the two above approaches could be to modify equation 5 where L is taken from the task policy while \mathbb{S} is taken for μ_σ , i.e.:

$$Kt^{[\epsilon, \mapsto \nu]}(\mu_\sigma | \mu) \triangleq \min_{\pi \in Opt_{Kt}^{[\mapsto \nu]}(\mu) \cap \mathcal{A}^{[\epsilon, \mapsto \nu]}(\mu_\sigma)} \mathbb{LS}^{[\mapsto \nu]}(\pi, \mu_\sigma) \quad (6)$$

This could be a more elaborate version of difficulty. Nonetheless, we must say that any of the above options is not proposed as a definitive policy that may give an intuitive value of difficulty for every possible task and instance. Our goal here is to show some of the ingredients about the notion of difficulty, and provide some useful references to construct one personalised version of instance difficulty for a given situation. Being more or less elaborate, we think that the principles must be the same. For instance, we emphasise that all of them are defined in terms of computational steps, so they actually measure algorithmic effort.

Of course, there will be occasions where the notion of difficulty for an instance is controversial. For instance, for a division task, imagine $2525/8527$, and now consider two different division algorithms π_1 and π_2 , which both are approximately equally efficient and short (same \mathbb{LS}). However, for π_1 we have that $7674/2558$ is solved very easily but $7674/2558$ takes much more steps for π_2 . Using any of the above definitions, the difficulty will depend on the reference machine that calculates length and complexity. Other more robust options, such as considering not only the minimum in equations 4 and 5 (or 6), have already been mentioned, and would lead to more complicated notions of difficulty. In any case, they could be thought as future work.

4.2 Task composition and decomposition

Now the question is to consider how we can put several tasks together. For instance, if we include μ_{num} and μ_{RPM} from Table 1 in the same test, does it make sense to aggregate the results? The first problem is that the aggregation of several responses that are not commensurate makes no sense (perhaps for one the responses go from 0 to 0.1 while for the other they go from 0.5 to 1, with very different distributions of results⁹). One alternative to a normalisation is to use a tolerance level for the tasks. This gives further justification to eq. 1, where \mathcal{A} was introduced. Given two tolerance levels for each task we can see whether this leads to similar or different difficulties for each task. For instance, if the difficulties are very different, then the task will be dominated by the easy one. In the previous example, the μ_{num} is much easier than the μ_{RPM} . By using different tolerance levels we can determine whether we want both tasks to have the same relevance or not. In fact, we do not really need to use different values of ϵ , as we can find a monotonic transformation of (one of) the responses such that the ϵ can be the same for both, leading to the same difficulty. Given any task, any monotonic transformation of the responses leads to another task such that there is another ϵ that leads to the same acceptability set.

Comparing the difficulties of the tasks for a response value is important to understand what the composition really means, but we have not defined what a composition is. Given two stochastic tasks, it does not make sense to make the union of the tasks, but rather to calculate a mixture. In particular, the composition of tasks μ_1 and μ_2 with weight $\alpha \in [0, 1]$, denoted by $\alpha\mu_1 \oplus (1 - \alpha)\mu_2$, is defined by a stochastic choice, using a biased coin (e.g., using α), between the two tasks. Note that this choice is made for each trial. It is easy to see that if both μ_1 and μ_2 are asynchronous-time stochastic tasks, this mixture also is.

Similar to composition we can talk about decomposition, which is just understood in a straightforward way. Basically, μ is decomposable into μ_1 and μ_2 if there is an α and two tasks μ_1 and μ_2 such that $\mu = \alpha\mu_1 \oplus (1 - \alpha)\mu_2$.

Now, it is interesting to have a short look at what happens with difficulty when two tasks are put together. Given a difficulty function h , we would like to see that if $h(\alpha\mu_1 \oplus (1 - \alpha)\mu_2) \approx \alpha h(\mu_1) + (1 - \alpha)h(\mu_2)$ then both tasks are related, and there is a common policy that takes advantage of some similarities. However, in order to make sense of this expression, we need to consider some values of α and fix a tolerance. With high tolerance the above will always be true as h is close to zero independently of the task. With intermediate tolerances, if the difficulties are not even, the policies for the composed task will invest more resources for the easiest ‘subtask’ and will neglect the most difficult ‘subtask’. For instance, if there is an easy policy for μ_1 achieving response 0.8, but for μ_2 the policies are much more difficult if the same level of response is aimed at, one can make do with a switch, use the easy policy for μ_1 and manage with an easy policy for μ_2 achieving response 0.4. If $\alpha = 0.5$ then we would have overall response of 0.6, which may be acceptable for intermediate tolerances. Finally, using low tolerances (or even 0) for the above expressions may have more meaning, as the policy must take into account both tasks. In fact, for tolerance 0 the value of α that is not 0 or 1 is not relevant.

In fact, there are some cases for which some relations can be established. Assume 0 tolerance, and imagine that for every $1 > \alpha > 0$ we have $h(\alpha\mu_1 \oplus (1 - \alpha)\mu_2) \approx \alpha h(\mu_1)$. If this is the case, it means that we require the same effort to find a policy for both tasks than for one alone. We can see that task μ_1 covers task μ_2 . In other words, the optimal policy for μ_1 works for μ_2 . Note that this does not mean that every policy for μ_1 works for μ_2 . Finally, if μ_1 covers μ_2 and vice versa, we can say that both tasks are equivalent.

We can also calculate a distance as $d(\mu_1, \mu_2) \triangleq 2h(0.5\mu_1 \oplus 0.5\mu_2) - h(\mu_1) - h(\mu_2)$. Clearly, if $\mu_1 = \mu_2$ then we have 0 distance. For tolerance 0 we also have that if μ_2 has difficulty close to 0 but μ_1 has a high

⁹One can normalise them by a cumulative distribution, again if we can figure out a population or distribution of policies.

difficulty h_1 , and both tasks are unrelated but can be distinguished without effort, then we have that the distance is h_1 .

These ideas and properties can be related to concepts such as (normalised) information distance [6, 61], especially the similarity of two tasks, with the appropriate caution, as here we are talking about interactive tasks and we are using the complexity of the policies and not the complexity of the description of the tasks. Two tasks with very similar (shortest) descriptions can have very different policies, and two tasks with very different (shortest) descriptions can have the same general policy. In the case of task composition and distance, we have seen that there are features for tasks that are fundamental, such as their magnitudes (which can be made even by the use of an appropriate tolerance or a monotonic function), their original difficulties and also whether both tasks can be distinguished easily (two similar tasks can be difficult to tell apart and putting them together could require a great extra effort).

Nonetheless, there are many questions we can analyse with this conceptualisation. For instance, how far can we decompose? That depends on how we decompose. For an infinite distribution (e.g., many stochastic tasks could be seen in this way), there are infinitely finer decompositions, each of them containing an infinite number of instances. But there are some other decompositions that will lead to tasks with very similar instances or even with just one instance. Let us consider the addition task μ_{add} with a soft geometrical distribution p on the numbers to be added. With tolerance 0, the optimal policy is given by a short and efficient policy to addition. We can decompose addition into μ_{add1} and μ_{add2} , where μ_{add1} contains all the summations $0 + x$, and μ_{add2} incorporates all the rest. Given the distribution p , we can find the α such that $\mu_{add} = \alpha\mu_{add1} \oplus (1 - \alpha)\mu_{add2}$. From this decomposition, we see that μ_{add2} will have the same difficulty, as the removal of summations $0 + x$ does not simplify the problem. However, μ_{add1} is simple now. But, interestingly, μ_{add2} still covers μ_{add1} . We can figure out many decompositions, such as additions with and without carrying. Also, as the task gives more relevance to short additions because of the geometrical distribution, we may decompose the task in many one-instance tasks and a few general tasks. In the one-instance tasks we would put simple additions such as $1 + 5$ that we would just rote learn (the optimal policy for these cases alone is just rote learn). In fact, it is quite likely that in order to improve the efficiency of the general policy for μ_{add} the policy includes some tricks to treat some particular cases or easy subsets. This can perfectly happen with some of the task difficulty functions seen before, such as $Kt^{\{\epsilon, 0 \rightarrow 1\}}(\mu)$. This is also consistent with many cognitive analyses of how humans perform addition (see, e.g., [5]). The use of decompositions can be useful to analyse many other cases. For instance, if we make a decomposition of π into π_1 and π_2 with a high α , and get that the difficulty of π_1 is low then it is quite likely that the original policy internally incorporates this separation. This can also happen with difficult instances (or subtasks) if tolerance is 0, as they can be incorporated in a rote-learning way. Also, it may be interesting to compare this (with tolerance 0) to the notion of instance difficulty seen in the previous section (which plays with levels of tolerance).

The opposite direction is if we think about how far we can reach by composing tasks. Again, we can compose tasks *ad eternum* without reaching more general tasks necessarily. The big question is whether we can analyse abilities with the use of compositions and difficulties. In other words, are there some tasks such that the acceptable policies for these tasks are frequently useful for many other tasks? That could be evaluated by looking what happens to a task μ_1 with a given difficulty h_1 if it is composed with any other task μ_2 of some task class. If the difficulty of the composed task remains constant (or increases very slightly), we can say that μ_1 covers μ_2 . Are there tasks that cover many other tasks? This is actually what psychometrics and artificial intelligence are trying to unveil. For instance, in psychometrics, we can define a task μ_1 with some selection of arithmetic operations and see that those who perform well on these operations have a good arithmetic ability. In our perspective, we would need to check whether the selection of operators that are evaluated (+, −, etc.) has some kind of optimal policy that does not help with the general problem. If this does not happen, then we could extrapolate (theoretically and not experimentally) that this task μ_1 covers a range of arithmetic tasks.

As more general we get with composition, things will become harder (but not impossible). Can we define a task for inductive ability and show that this will cover every other pure inductive task? Or that it will be helpful for other tasks featuring inductive abilities? In artificial intelligence, this is usually the set of general techniques (in vision, pattern recognition, natural language processing) that are reused again and again in different applications. An ultimate question is whether there is a general task such that it is useful for every other task (like general intelligence or the g factor), especially in cases with many trials (e.g., using

$Kt^{[\epsilon, \mapsto \infty]}(\mu)$). The view of Table 2 and some of the optimal policies being $\pi_{find-LS-best}$ suggest that for very ‘large’ tasks, these general, meta-search, algorithms may be good policies for these tasks.

All of the above is just some directions that should be analysed in detail with separate research pieces. In practice, even if difficulty functions such as $Kt^{[\epsilon, \mapsto 1]}(\mu)$ may be computationally expensive to calculate for many tasks, these set a conceptual framework to analyse many of these questions. For instance, the notion of task breadth can be analysed in this context. There have been several (informal) approaches or expressions of relevance of task breadth [14, 49] or the notion of intellectual breadth (though applied to an agent [13] and not to a task). Some of them are relative to other tasks or to humans, such as the one suggested (but not fully developed) with the Turing Ratio [46]. With the observation of how difficulty changes with composition and decomposition we could try to give a proper formalisation of task breadth or, alternatively, we may reach the conclusion that task breadth is not a meaningful notion.

Finally, the additivity of difficulty with composition could be analysed, and compared to other kinds of combination. Imagine two tasks μ_1 and μ_2 that are put together (sequentially) as a new joint μ , and an observation signals whether the agent is performing well for each of the two parts. A policy could be ‘Identify parts. Find the policy for the first part and the second part independently’. That means that the policy could do separate searches. For instance, in the worst case (or using a Levin search) we could have $2^{L(\pi_1)} + 2^{L(\pi_2)}$ possible choices instead of $2^{L(\pi_1) + L(\pi_2)}$, where π_1 and π_2 are the partial programs that solve the partial subtasks. A concatenation of tasks is very different from a composition, but if agents have memory, we could find cases where there are connections.

4.3 Agent response curves

One of the usefulness of difficulty is the analysis of agents according to how they behave in terms of the difficulty of the problem. This can be done with the so-called agent response curves, introduced in [25] following the notion of item response curves in psychometrics. Let us see briefly how these curves can be defined for tasks or task instances.

We first define $\mathbb{A}^{[\epsilon, \mapsto \nu]}(\pi, \mu) \triangleq 1$ if $\pi \in \mathcal{A}^{[\epsilon, \mapsto \nu]}(\mu)$ and 0 otherwise. A task class is defined as a pair $\langle M, p_M \rangle$, where M is a set of tasks or task instances, and p_M is a distribution. Note that with this definition, task classes are stochastic tasks (but not all stochastic tasks can be seen as classes). We also consider a difficulty function h (over tasks, or over task instances relative to the overall task).

We can group those of the same difficulty:

$$\Psi_h^{[\epsilon, \mapsto \nu]}(\pi, M, p_M) \triangleq \sum_{\mu \in M, h^{[\epsilon, \mapsto \nu]}(\mu) = h} p_M(\mu|h) \cdot \mathbb{A}^{[\epsilon, \mapsto \nu]}(\pi, \mu) \quad (7)$$

If we represent Ψ_h on the y -axis versus h on the x -axis we have a so-called agent response curve, as shown in Figure 2. In order to have a nice view of the figure, we need to investigate how the points are derived. Do we have elements of M with the same value of h ? Otherwise, the values of the y -axis would all be either 0 or 1. In order to observe values between them we must have several elements in M with the same of h . If h is a continuous function and this is not the case, we can group h by intervals. This can also be done for convenience if there are no elements for some regions of h , so that we get a ‘continuous’ curve without empty regions.

The important thing, however, is how it works. Here we have three particular cases:

- If the elements of M are stochastic tasks each and we use a non-relative version of difficulty, the curve may have a look very much the same as Figure 2. For some of the difficulty functions seen in the previous sections (K in particular¹⁰), it can be shown that for every agent there is an h such that above it, Ψ_h is zero (this h is actually the length of the algorithm of the agent).
- If the elements of M are variants of the same task by varying the value of ϵ then we have that the curve is a non-increasing step function, where the leap of the step is located at the difficulty h of the variant of the task such that the tolerance equals the achievement of agent π . This curve is of course not very

¹⁰For Kt this cannot be proved in general unless we include the computational steps the policy takes into R . However, this would go against a behavioural evaluation. Nonetheless, for those tasks for which there is some relevance of time to the R and assuming non-infinite speed of the agent, we can show that this is bounded.

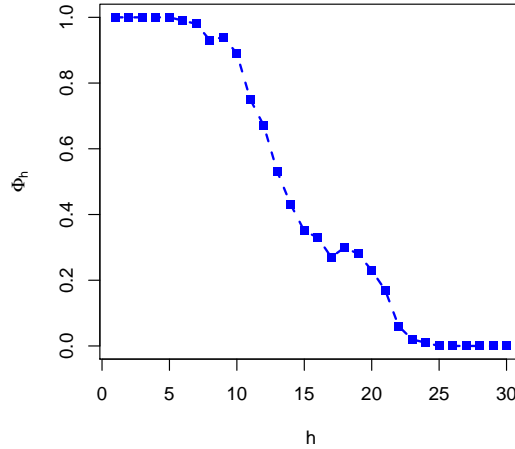


Figure 2: An agent response curve.

informative for one agent. However, this could be interesting if we aggregate these ‘step’ curves for a population of agents.

- If the elements of M are instances and we use a relative version of difficulty, assuming that the function is defined for all instances, we may have non-zero values for arbitrary high values of h , even for very simple agents (for instance, if an agent only solves a single instance, but this instance is difficult). At least this h can be bounded by the h of the task (if it exists for 0 tolerance).

For the last case, we can see that if there are many instances with the same difficulty (or we aggregate values of h in intervals), then we are considering an average of results for many instances and the shape will be mostly non-decreasing, like the one shown in Figure 2.

5 Difficulty as Levin search with stochastic verification

We decided to associate difficulty to the *smallest number of computational steps such that we get an acceptable policy for a given tolerance ϵ* . This depends on how many alternative algorithms we need to try before we find the right one and how much time we require in order to discard the bad ones and confirm the correct one. This boils down to a measure of difficulty that depends on how many options need to be explored and the time that each of them takes. Their product will give an upper bound of the number of computational steps to find the best acceptable policy to a problem, i.e., its difficulty.

In previous sections we considered the length of the policy and the logarithm of its computational time through their combination \mathbb{LS} , which finally led to the function $Kt^{[\epsilon, \mapsto \nu]}(\mu)$. As we argued, this is given by the realisation that in order to find a policy of length $L(\pi)$ we have to try approximately $2^{L(\pi)}$ algorithms if we enumerate programs from small to large (this is basically what Levin search does, as we will see below). Considering that we can also gradually increase the computational steps that we devote for each of them, we get $2^{L(\pi)} \cdot \mathbb{S}(\pi, \mu)$, whose logarithm is represented by Kt . This is why we say that the unit of Kt is logarithm of computational steps¹¹.

If we try to extend this notion to tasks, the first, and perhaps most obvious and important difference with traditional Levin’s universal search is that tasks are stochastic. Consequently, several trials may be needed for discarding a bad policy and the verification of a good one. This is specially the case when the response can have a high variance. Even a good policy can give bad results eventually, and we cannot discard a good policy just because it fails for one trial. We require repetitions, i.e., more trials, to know whether the policy is good for the whole task on average or not. Intuitively, a pair of task and policy with low variability in the response (results) will be easier to be verified than another where results behave more stochastically. For

¹¹[47] says “this allows time to be measured in bits”, but I think that this is misleading, as there is more information involved.

Feature	Kind of difficulty it represents	Notation	Depends on
Information content (size) of π	Transmission (language or coding)	$L(\pi)$	-
Execution steps of π	Demonstration	$\mathbb{S}(\pi, \mu)$	-
Expected value of response of π	-	$\mathbb{R}(\pi, \mu)$	-
True variance of response of π	-	$\text{Var}[R(\pi, \mu)]$	-
Verification trials of π	-	$\mathbb{B}(\pi, \mu)$	\mathbb{R} and $\text{Var}[R(\pi, \mu)]$
Finding effort steps of π	Finding (trivial or no verification)	$\mathbb{LS}(\pi, \mu)$	L and \mathbb{S}
Verification steps of π	-	$\mathbb{W}(\pi, \mu)$	\mathbb{S} and \mathbb{B}
Total effort steps of π	Search (with target)	$\mathbb{F}(\pi, \mu)$	L and \mathbb{W}

Table 3: Different features of a policy π given a task μ .

instance, consider two possible policies: a and b . Problem μ_1 gets response 0 or 1 (with equal probability) for policy a whereas it gives a constant value of 0 for policy b . This case is more difficult to find and verify than another problem μ_2 where policy a consistently gets a constant response of 0.5. Note that in both cases, the expected response for policies a and b is 0.5 and 0 respectively, but it is intuitive to think that the first case is more difficult to find (actually because it is harder to verify).

The second difference with classical Levin search is that the search algorithm goes through several trials, and it is not clear that the agent can interrupt the trial if a policy does not look promising. Nonetheless, we can consider that the search algorithm can also do some **sleep** operations so that basically nothing is done until a new algorithm can be tried for the following trials.

The third difference is that we can think about a Levin search with memory, as some of the observations on previous trials may be crucial (whereas Levin search is basically a blind search). So we need that the policies that are tried could also be search procedures over several trials. That means that Levin search actually becomes a metasearch, which considers all possible search procedures, ordered by size and resources¹². Only in this way we can properly give an intuitive measure of difficulty for μ_{srote} and μ_{lrote} in Table 2.

This is just a realisation that for interactive stochastic scenarios, verification is not just one execution, but many if things are stochastic, because there is noise, the systems are not foolproof, etc. In a way, we are looking for more general and robust searches. This view is not very different to many evolutionary processes that have tried many policies in a world that is basically stochastic.

Another important thing is that in order to calculate the computational steps of a search, this search must stop at some point and say that the good policy has been found. However, as tasks are stochastic, we can never have complete certainty that a good policy has been found. An option is to consider a confidence level, such that the search invests as fewer computational steps as possible to have a degree of confidence $1 - \delta$ of having found an ϵ -acceptable policy. This clearly resembles a PAC (probably approximate correct) scenario [?].

Before starting, Table 3 summarises some of the notations we will use. We must also bear in mind that we are focussing on a view of difficulty when the policy is found by search (be it “intellectual”, “evolutionary” or “cultural”, as Turing distinguished [57]). However, the table also shows that there are other ways of acquiring a policy (by transmission, by demonstration or by search).

5.1 Levin universal search for stochastic tasks and/or policies

Levin’s universal search has very interesting properties, as any inversion problem can be solved optimally (except for a multiplicative constant) [44, pp. 577–580]. It is related (and with approximately similar properties) to the SIMPLE search algorithm in [44, pp. 579], but with the advantage that the execution of programs does not need threads or traces to be kept in order to resume previously explored program executions (at the cost of repeating part of previous executions). The important thing is how they relate the length of a program with their execution (and verification) time.

The traditional Levin’s universal search is defined as follows:

¹²There are some variants and adaptations of Levin search for interactive scenarios and MDPs [33, 52, 34, 53, 51]. Here it is not our goal to find a search that is useful to design intelligent agents but to find some expressions that help us refine our definition of *task* difficulty.

Definition 1. *Levin's universal search.* Given a string x and a universal prefix-free Turing machine U , for which programs can be enumerated, we conduct several phases, starting from phase 1. For phase i , we execute all possible programs p with $L(p) \leq i$ for at most $s_i = 2^{i-L(p)}$ steps each, including in this limit s_i the steps¹³ needed to verify whether $U(p) = x$. Once we find and verify the first successful policy the search is stopped. Otherwise we continue until we complete the phase and then to a next stage $i + 1$.

The number of steps to execute p and verify that it produces x or not¹⁴ is denoted by $W(p, x)$. We can determine an upper bound of the total number of steps taken by this procedure. While one could expect that this is $s \leq (k - 1)2^{k+1} + 2$, this is significantly reduced by the use of prefix-free programs, so Kraft inequality can be used, having:

Theorem 1. ([44, pp. 580, claim 7.5.1]) *The number of steps s taken by Levin search given by definition 1 is bounded by:*

$$s \leq 2^{k+1}$$

where

$$k = L(p) + \log(W(p, x)) \quad (8)$$

and p is the first program that meets the stop condition.

Even if we can use Kraft inequality and we get a much tighter upper bound, it seems that this bound is still rather loose, as many programs may stop before the allotted stops. However, as we can think of UTMs for which all programs of size lower than a constant may have the properties that we would like, this upper bound cannot be made lower in general (although some systems can exploit it for some other UTMs).

The use of k as for equation 8 in theorem 1 suggests that we use this expression as a standalone expression:

$$\log F(p, x) \triangleq \log 2^{L(p)} \cdot W(p, x) = L(p) + \log(W(p, x)) \quad (9)$$

As we are considering non-probabilistic programs and an identification problem (and not really an inversion problem for any given partial recursive function), we do not need parameter x here.

$$\log F(p) \triangleq \log 2^{L(p)} \cdot W(p, U(p)) = L(p) + \log(W(p, U(p)))$$

According to the above process, it is easy to see that using the above procedure the first returned program that outputs x will be one that minimises: $Kt(x) \triangleq \min_{p: U(p)=x} \log F(p)$.

Levin's search assumes that there is a fast way of verifying policies. Now in the case of interactive stochastic systems with a response function, the procedure cannot just verify that the policy is correct by executing it once. Also, for each execution of the same program the number of steps can be different. How can we adapt universal search to this situation?

Definition 2. *Levin's universal search for stochastic tasks and policies.* Given a problem x and a universal prefix-free Turing machine U , for which policies to x can be enumerated, we conduct several phases, starting from phase 1. For phase i , we execute all possible programs p with $L(p) \leq i$ for at most $s_i = 2^{i-L(p)}$ steps each. In these steps we include the steps required to execute p several times to consider that p is a policy for x (within the allotted number of steps). As soon as the policy is deemed to be incorrect or the allotted number of steps is exhausted, we try the next program. On the contrary, if the policy is verified the search is stopped. While a policy is not found we continue until we complete the phase and then to a next stage $i + 1$.

The number of verification steps now¹⁵ depend on stochastic executions and may vary (that is why we denote them by \mathbb{W}). And similarly, we get the equation for effort equal to eq. 9. In this case, we cannot get rid of x in the definition, as p may be stochastic and \mathbb{W} is understood as an expected value.

¹³In order to verify a string we need to compare bit by bit with x . Note that this is not going to be constant. In the worst case, this takes $c \cdot L(x)$ steps, with c being the computational steps per bit verification of a program that goes bit by bit over x . However, on average (assuming a 0.5 probability that a random program guesses each bit right), we have that the expected value is $\sum_{i=1}^{L(x)} i2^{-i}$, which converges to 2, so we will have $c \cdot 2$ steps on average. This is the reason why this verification part is often ignored for identification problems.

¹⁴In this case, at the first moment that the string produced by p does not match x the verification is stopped.

¹⁵Actually, the number of verification steps was also an expected value, as depends on the differences between the reference string and the output of the program.

5.2 ‘PAC’ verification for stochastic tasks

Now we are going to adapt this to stochastic tasks. We again realise that for a stochastic system we can never be 100% sure of a policy, because even after a million successes we can have a failure. The second thing is that for stochastic systems it may be unreasonable to expect maximum or perfect result. In fact, by using any statistical test about whether we have reached the maximum value, we cannot have any degree of certainty (even the slimmest) of having this maximum value, as our average so far will never be above the maximum value, which is necessary for statistical significance.

We could think about using *one* slack parameter. Given a series of n runs, we can calculate the average \hat{r} and the standard deviation σ of the results. For instance, we can get the standard error by just $SE = \frac{\sigma}{\sqrt{n}}$ and set a limit on it. However, if we do this, we see that it depends on the magnitude. For instance, a stochastic process alternating between 0 and 1 will have higher σ than if it alternates between 0.4 and 0.6, just by scaling, even if the verification cost (of knowing whether it is above, e.g., 0.7, or not) looks the same.

Instead, we are going to consider two parameters. We want the search procedure to find a policy with a confidence level δ , i.e., $Pr(\pi \text{ solves } \mu) \geq 1 - \delta$. As mentioned above, if we consider the best possible result (i.e., 1) to acknowledge that this is solved, then even with high values of δ this will never be achieved. So the second thing is that if we consider a utility, response or result function R , we must set that the difference with respect to the best policy is lower than a given error ϵ . If we denote the best possible average result (for an infinite number of runs) as r^* (note that r^* can be lower than 1), we consider that a series of runs is a sufficient verification for a probably approximate correct (PAC) policy π for μ when:

$$Pr(r^* - \hat{r} \leq \epsilon) \geq 1 - \delta \quad (10)$$

with \hat{r} being the average of the results of the trials (runs) so far. As $r^* - \hat{r} \leq \epsilon$ is the same as $\hat{r} \geq r^* - \epsilon$, sometimes $r^* - \epsilon$ will be referred to as the ‘threshold’ or ‘target’. For instance, if the achievable maximum is 0.9 and $\epsilon = 0.15$ then our threshold is 0.75.

Now we are ready to give an expression for the verification steps for a given problem μ and a policy π . Namely, the number of verification steps $\mathbb{W}^{[\epsilon, \delta]}(\pi, \mu)$ is defined as the expected value of the parameter s returned by VERIFYGEN (Algorithm 1) for $s_{max} = \infty$.

Algorithm 1 VERIFICATION ALGORITHM (GENERIC)

```

1: function VERIFYGEN( $\pi, \mu, \epsilon, \delta, s_{max}$ )                                 $\triangleright s_{max}$  is the number of allowed steps
2:    $j \leftarrow 1$ 
3:    $s \leftarrow 0$ 
4:    $m_\pi \leftarrow \emptyset$                                                  $\triangleright$  The algorithm  $\pi$  can keep memory between trials. Initially empty.
5:   repeat
6:      $\langle r_j, s_j, m_\pi \rangle \leftarrow \text{Run}(\pi, m_\pi, \mu, s_{max} - s)$        $\triangleright$  One trial with at most the  $s_{max} - s$  remaining steps
7:                                                                                    $\triangleright$  Run returns response and used steps
8:      $s \leftarrow s + s_j$                                                  $\triangleright$  Accumulate steps
9:      $r \leftarrow r + r_j$                                                  $\triangleright$  Accumulate response
10:     $\hat{r} \leftarrow \frac{r}{j}$                                                  $\triangleright$  Average response
11:     $p \leftarrow Pr(r^* - \hat{r} \leq \epsilon)$                                  $\triangleright$  We calculate this probability in some way
12:    if  $p \geq 1 - \delta$  then return  $\langle \text{TRUE}, s \rangle$                          $\triangleright$  Stop because it is verified
13:    else if  $p \leq \delta$  then return  $\langle \text{FALSE}, s \rangle$                      $\triangleright$  Stop because it is rejected
14:    end if
15:     $j \leftarrow j + 1$ 
16:  until  $s \geq s_{max}$ 
17:  return  $\langle \text{FALSE}, s \rangle$ 
18: end function

```

Note that we require r^* , which is defined as the highest expected response of any resource-bounded policy (in LS). If this is not known, we can assume $r^* = 1$, as in previous sections.

Algorithm 1, if using an appropriate estimation of the probability for stopping in each iteration, may have a tendency of stopping prematurely because each iteration depends on the previous ones. Actually,

especially in the beginning, this is vulnerable to spurious results and very bad estimations of the mean and the variance of the response. This is basically the problem of (large-scale) multiple testing. One strong correction is Bonferroni method, where the confidence per test is modified to: $\delta' = \delta/n$, but as n are the repetitions so far, it would be an incremental test.

Also, if used with definition 2, this problem is exacerbated. As we evaluate about 2^i programs in each phase (actually slightly less than this because it is a prefix code), we have that there can be cases that are just accepted (the condition $Pr(r^* - \hat{r} \leq \epsilon) \geq 1 - \delta$ becomes true) by chance. Note that a rejection by chance is not so problematic, as the same program will be evaluated in the following phase again.

Given all the considerations above, we realise that it is going to be very difficult to find the exact statistical criteria to stop by acceptance and rejection. In what follows, we just propose an approximation to the upper limit with the goal of recognising the difference in difficulty between finding an acceptable policy for stochastic problems with high margin ($\hat{r} + \epsilon - r^*$) and small standard deviation and those with tighter margins and higher standard deviations.

First, we are going to assume that all runs take the same number of steps (a strong assumption, but let us remind that this is an upper limit¹⁶), so the verification cost above could be approximated by

$$\widehat{\mathbb{W}}^{[\epsilon, \delta]}(\pi, \mu) \triangleq \mathbb{S}(\pi, \mu) \cdot \mathbb{B}^{[\epsilon, \delta]}(\pi, \mu) \quad (11)$$

i.e., the product of the expected number of steps times the expected number of verification bids (iterations needed of the loop of Algorithm 1). With this, we focus on calculating the number of bids of the policy until we verify it is acceptable or not.

The number of bids can be estimated if we have the mean and the standard deviation of the response for a series of runs. If the conditions of the central limit theorem held, we could consider that the results of the bids would be normally distributed. In our case, the trials are not independent (neither are they ergodic) if we consider that the algorithm has memory between the trials, but nevertheless we will make this assumption, as, in general, we cannot make any further assumption about the distribution of the responses. As a result we can use the confidence level given by the normal distribution. The confidence interval is given by $\hat{r} - \frac{|z_{\delta/2}|\sigma}{\sqrt{n}}, \hat{r} + \frac{|z_{\delta/2}|\sigma}{\sqrt{n}}$. Where $z_{\delta/2}$ is the standard normal quantile. For instance, for $\delta = 0.05$, we have $|z_{0.025}| = 1.96$. We want this interval width w to be at most twice the margin over the threshold $r^* - \epsilon - \hat{r}$. So, $w \leq 2(\hat{r} + \epsilon - r^*)$. As $w = 2\frac{|z_{\delta/2}|\sigma}{\sqrt{n}}$, we have: $2\frac{|z_{\delta/2}|\sigma}{\sqrt{n}} \leq 2(\hat{r} + \epsilon - r^*)$. By isolating n we have:

$$n \geq \frac{|z_{\delta/2}|^2 \sigma^2}{(\hat{r} + \epsilon - r^*)^2} \quad (12)$$

Note that the above formula is infinite when $r^* - \epsilon = \hat{r}$, i.e., when we have that the policy reaches the threshold exactly. We cannot verify it is above the threshold for any confidence level.

In order to apply the above expression we need the variance σ^2 . If we just have one run, this is undefined, and for very few runs this is going to be poorly estimated. Many approaches to the estimation of a population mean with unknown σ^2 are based on a pilot or prior study (let us say we try 30 repetitions) and then derive n using the normal distribution and then use this for a Student's t distribution. Instead of this, we are going to take an iterative approach where we update the mean and standard deviation after each repetition. The problem, of course, happens with the first iterations. One approach we will take is to consider the maximum standard deviation as a start (as a kind of Laplace correction). As we assume that the response R is between 0 and 1, we will consider¹⁷ two fabricated repetitions with responses 0 and 1. With this, our start sample standard deviation will be high from the beginning and a minimum of iterations will always take place.

Algorithm 2 is a modification of Algorithm 1 where we use eq. 12.

Finally, we modify definition 2 by considering that when we find a verified policy we repeat the verification again with some extra repetitions (for instance, $n = 30$, so that the used normal distribution is a more sustainable assumption). Note that this extra verification will be performed just very occasionally, so this will not significantly affect the number of steps taken by the modified Levin search. With all this, the modified version is as follows:

¹⁶For instance, if $\epsilon = 0.5$ and all bad policies have response 0.499999 and there is only one good policy with response 0.55, we will require many repetitions to discard the bad policies, until we find and verify the good policy.

¹⁷Other options exist, such as deriving some initial values depending on the threshold.

Algorithm 2 VERIFICATION ALGORITHM (NORMALITY)

```

1: function VERIFYNORM( $\pi, \mu, \epsilon, \delta, s_{max}$ )                                 $\triangleright s_{max}$  is the number of allowed steps
2:    $j \leftarrow 3$                                                           $\triangleright$  We consider two first response with high variance
3:    $r \leftarrow 0 + 1$                                                       $\triangleright$  One with value 0 and the other with value 1
4:    $s \leftarrow 0$ 
5:    $m_\pi \leftarrow \emptyset$                                                 $\triangleright$  The algorithm  $\pi$  can keep memory between trials. Initially empty.
6:   repeat
7:      $\langle r_j, s_j, m_\pi \rangle \leftarrow \text{Run}(\pi, m_\pi, \mu, s_{max} - s)$        $\triangleright$  One trial with at most
8:      $\triangleright$  the  $s_{max} - s$  remaining steps  $\text{Run}$  returns response and used steps
9:      $s \leftarrow s + s_j$                                                   $\triangleright$  Accumulate steps
10:     $r \leftarrow r + r_j$                                                   $\triangleright$  Accumulate response
11:     $\hat{r} \leftarrow \frac{r}{j}$                                                   $\triangleright$  Average response
12:     $\hat{\sigma}^2 \leftarrow \text{Var}[r_1 \dots r_j]$                                 $\triangleright$  Variance estimation
13:     $n_0 \leftarrow \frac{|z_{\delta/2}|^2 \hat{\sigma}^2}{(\hat{r} + \epsilon - r^*)^2}$ 
14:    if  $j \geq n_0$  then
15:      if  $\hat{r} > r^* - \epsilon$  then return  $\langle \text{TRUE}, s \rangle$                      $\triangleright$  Stop because it is verified
16:      else return  $\langle \text{FALSE}, s \rangle$                                         $\triangleright$  Stop because it is rejected
17:      end if
18:    end if
19:     $j \leftarrow j + 1$ 
20:  until  $s \geq s_{max}$ 
21:  return  $\langle \text{FALSE}, s \rangle$ 
22: end function

```

Definition 3. *Levin’s universal search for stochastic tasks and policies with given tolerance ϵ , confidence level $1 - \delta$, and maximum response reference r^* . Given a task μ for which policies can be enumerated. We conduct several phases, starting from phase 1. For phase i , we execute all possible policies π with $L(\pi) \leq i$ for $s_i = 2^{i-L(\pi)}$ steps each. We call function $\text{VERIFYNORM}(\pi, \mu, \epsilon, \delta, s_{max})$ in Algorithm 2 with $s_{max} = s_i$. While an acceptable policy is not found we continue until we complete the phase and then to a next stage $i+1$. If an acceptable policy is found, some extra trials are performed before stopping the search for confirmation.*

Theorem 2. *For every μ and $\epsilon, \delta > 0$, if a maximum r^* exists achievable by a computable policy¹⁸ and it is given, then definition 3 conducts a finite search.*

Proof. As r^* is defined as the highest expected response for a resource-bounded policy π^* (in \mathbb{L}) and it exists, then there is a number of phases where π^* has already been found and there are enough steps such that \hat{r} is becoming as closer to r^* as needed such that $\hat{r} + \epsilon - r^*$ is positive and sufficiently close to ϵ such that is verified $\Pr(r^* - \hat{r} \leq \epsilon) \geq 1 - \delta$. Note that as results are bounded between 0 and 1 the highest variability is $\sigma^2 = 1/4$, so we have that $n \sim \frac{|z_{\delta/2}|^2 \sigma^2}{(\epsilon)^2}$ is bounded. \square

Note that if instead of r^* we give a higher value that, subtracted the error tolerance, cannot be attained, then the search is not bounded. Also note that in any case there can be a very simple policy equal to $r^* - \epsilon$ and will never be found.

Definition 3 is conceived to find the optimal policy, and it is not parametrised to calculate how long the search is to *discard* non-optimal policies. Actually, what we do is to use the approximation (i.e., equation 11) into another approximation for any possible π , assuming that π were the best policy.

In the end, what we want is to have a term that accounts for the variability of computational steps given by the variance of the response and its proximity to the threshold, as both things make verification more difficult. This is finally calculated as:

$$\mathbb{B}^{[\epsilon, \delta]}(\pi, \mu) \triangleq \frac{|z_{\delta/2}|^2 \text{Var}[R(\pi, \mu)]}{(\mathbb{R}(\pi, \mu) + \epsilon - r^*)^2} \quad (13)$$

¹⁸It could not exist if there is a never-ending series of programs requiring, e.g., more time to get a slightly better policy. It exists if there is a limit of steps (not time) with the interaction with the environment.

For both $\text{Var}[R(\pi, \mu)]$ and $\mathbb{R}(\pi, \mu)$ we consider that we include two extra responses as a start, as done in Algorithm 2.

And now the effort (eq. 9) is rewritten as:

$$\log \mathbb{F}^{[\epsilon, \delta]}(\pi, \mu) \triangleq \log(2^{L(\pi)} \cdot \widehat{\mathbb{W}}^{[\epsilon, \delta]}(\pi, \mu)) = L(\pi) + \log \widehat{\mathbb{W}}^{[\epsilon, \delta]}(\pi, \mu) \quad (14)$$

For clarity, we can expand what \mathbb{F} is by using the definition $\widehat{\mathbb{W}}$ from eq. 11 and taking the bids from eq. 13 as:

$$\log \mathbb{F}^{[\epsilon, \delta]}(\pi, \mu) = L(\pi) + \log \mathbb{S}(\pi, \mu) \cdot \mathbb{B}^{[\epsilon, \delta]}(\pi, \mu) = L(\pi) + \log \mathbb{S}(\pi, \mu) + \log \mathbb{B}^{[\epsilon, \delta]}(\pi, \mu) \quad (15)$$

It is a good question to determine whether \mathbb{S} and \mathbb{B} have comparable magnitudes. If the policies take thousands of steps and the number of repetitions is in the order of dozens or hundreds, then the variability of the responses will not be very important, and the difficulty will be dominated by L and \mathbb{S} . However, this depends on the task; there are of course cases for which \mathbb{B} can be very relevant.

From here, we can finally define a measure of difficulty that accounts for all the issues that affect the search of the policy for a stochastic task:

$$h^{[\epsilon, \delta]}(\mu) \triangleq \min_{\pi} \log \mathbb{F}^{[\epsilon, \delta]}(\pi, \mu) \quad (16)$$

It is important to compare this definition with those in section 3.2 and Table 1. Algorithm 2 considers that the algorithm has memory between tasks, so we are really extending $Kt^{[\epsilon, \rightarrow \nu]}(\mu)$ in section 3.2—but it can be modified easily without memory. The good thing is that now we do not need to specify ν any more, as the number of trials is given by Levin’s search itself. This takes some of the (best) cases from the two columns of Table 1 with Kt .

5.3 Interpretation and use

Does the approximation in equations 15 and 16 work properly? In order to get more insight about how it works we are going to see some figurative examples and see the values that would result, in order to see the effect of \mathbb{B} in the new formula of difficulty. This is shown in Table 4 (all cases consider $r^* = 1$). As we see from the results, there are cases where \mathbb{B} can be large and have effect on $\log(\mathbb{F})$.

While the use of \mathbb{B} includes an extra complication to the notion of difficulty, it does not add any significant additional cost in its computation, as for \mathbb{S} we need to execute the optimal policy many times. Nonetheless, the most difficult part of the estimation of difficulty is finding the optimal policy π^* .

The previous sections can be analysed in terms of whether they lead to bounded difficulty functions. It seems that for the target case (section 5.2) we have that if $r^* = 1$, the difficulty function is unbounded, but otherwise it can be bounded.

The number of repetitions is related to effort. We have argued that this is an upper approximation, but it can be much lower in many occasions. For instance, if an environment gives rewards 0 and 0.6 uniformly randomly independently of the action, so the expected response is 0.3, a threshold on 0.29999999 will lead to a high W but there is nothing to choose, as all policies behave the same, and whatever the agent does would be the same. All other policies get the same result, so there is no need for effort for discarding hypotheses, or dangers in guessing a wrong policy, etc. This is related to unquestionability, as whether there are competing programs of similar complexity is relevant, as in [29, 15]. However, a Levin search (or a real agent) does not have this information, so all the verification effort has to be done anyway.

6 Conclusions

As we have mentioned during this paper, the notion of task is common in AI evaluation, in cognition and also in human evaluation. However, a general formalisation, their arrangement and, most especially, their difficulty has not been addressed with earnest determination. Of course, with this resolution of being general, we have left some other more comfortable approaches, such as MDP and other formalisations in AI. Our main goal was difficulty, as we have seen that this is central to many of the other questions. Difficulty is seen as computational steps of a Levin search, but this search has to be modified to cover stochastic behaviours.

responses	$1 - \epsilon$	$1 - \delta$	$L(\pi^*)$	$\mathbb{S}(\pi^*, \mu)$	\mathbb{R}	σ	\mathbb{B} with $\hat{\sigma}$	\mathbb{B} with σ	$\log(\mathbb{F})$ with $\hat{\sigma}$
anything	anything	1			-	-	∞	∞	
1	1	anything			1	0	49	NaN	
1	0.99	0.95			1	0	197	0	
0*	0.01	0.95			0	0	197	0	
1	0.3	0.95			1	0	2	0	
0.3	0.3	0.95			0.3	0	∞	0	
{0.002, 0.018}	0.01	0.95			0.019	0.008	∞	∞	
{0.002, 0.018}	0.009	0.95			0.019	0.008	28	62	
1 one in 100	0.009	0.95			0.01	0.0995	7600	9507	
N(0.01, 0.001)	0.009	0.95			0.01	0.001	26	9	
0.01	0.009	0.95			0.01	0	649	0	
{0, 1}	0.45	0.95			0.5	0.5	98	97	
0.5	0.45	0.95			0.5	0	14	0	
0.55	0.5	0.95			0.55	0	16	0	
0.45*	0.5	0.95			0.45	0	16	0	
0.5	0.3	0.95			0.5	0	4	0	
0.5	0.3	0.9			0.5	0	3	0	
0.5	0.3	0.99			0.5	0	5	0	
1	0.5	0.95	10	200	1	0	4	0	$10 + \log(800) = 19.6$
0.51	0.5	0.95	5	100	0.51	0	93	0	$5 + \log(9300) = 18.2$
0.51	0.5	0.95	7	20	0.51	0	93	0	$7 + \log(1860) = 17.0$
0.51	0.5	0.95	10	200	0.51	0	93	0	$10 + \log(18600) = 24.2$
anything	0	anything	-	-	-	-	0	0	

Table 4: Examples of stochastic tasks. We are assuming $r^* = 1$ (see second column). We figure out an optimal policy π^* and see what value for \mathbb{F} would result (note that we are not doing an actual Levin search here). All estimations are using smoothing by the inclusion of a result 0 and 1 at the beginning of the results vector, as in Algorithm 2. We use 1,000 trials to calculate the true expected response and the true variance. \mathbb{R} and σ are shown without the smoothing. The expected number of bids (\mathbb{B} with $\hat{\sigma}$) is calculated incrementally until the number of repetitions needed to calculate a value of n (the repetitions) is lower than the current iteration, as if the variance were approximated incrementally. The same calculation with the perfect value of σ is represented in the next column: (\mathbb{B} with σ). Note that \mathbb{W} is approximated here as a product of expected values, as it is actually the expected value of an algorithm using many runs. The asterisks in the first column represent that these are cases played with policies that are rejected (just for comparison).

Nonetheless, we have been able to find an expression in terms of the best policy for the task. These ideas are an evolution and continuation of early notions of task and difficulty in [25] and [21] respectively.

There have been some early approaches where the role of Kt has been explored for different kinds of optimisation or inference problems [16, 26, 30, 18, 17]. The disposition and arrangement of tasks was discussed in [19], as well as the notion of task or agent breadth [46, 14, 49], and the distinction between specific and general [31]. The notions introduced in this paper, and the expression for difficulty can be useful to reunderstand some of the recent contributions in the evaluation of intelligence [41, 32, 22, 20, 37, 24, 11, 27, 28, 35, 36, 39, 9, 42, 10, 23, 38].

The relevance of verification in difficulty has usually been associated with deduction. However, some works have incorporated it as well in other inference problems, such as induction and optimisation, using Levin’s Kt [17, 47, 2].

We can briefly mention some issues that we have not fully developed here. First, we limit difficulty to the complexity of the best policy. However, the notion would be more robust if we considered more policies and their aggregation using a (universal) distribution. This is in principle possible, but would make the expression more convoluted and the notions of composition and decomposition trickier to analyse. A second issue is that in the second part of the paper we have not discussed the value of ν as in $Kt^{[\epsilon, \mapsto \nu]}(\mu)$ in section 3.2, because it is said to be given by the Levin’s search. However, this could be further investigated. Many other things could be explored, especially around the notions of composition and decomposition, task instance and agent response curves. Also, while our use of ‘PAC’ is just superficially related to PAC learning, we may have a closer look as this, in particular in the context of PAC reinforcement learning [55].

References

- [1] José Z Abramson, Victoria Hernández-Lloreda, Josep Call, and Fernando Colmenares. Relative quantity judgments in south american sea lions (*otaria flavescens*). *Animal cognition*, 14(5):695–706, 2011. 3
- [2] T. Alpcan, T. Everitt, and M. Hutter. Can we measure the difficulty of an optimization problem? *IEEE Information Theory Workshop (ITW)*, 2014. 27
- [3] I. Arel, D. C. Rose, and T. P. Karnowski. Deep machine learning - a new frontier in artificial intelligence research. *Computational Intelligence Magazine, IEEE*, 5(4):13–18, 2010. 8
- [4] D. Ashlock and J. Schonfeld. Evolution for automatic assessment of the difficulty of sokoban boards. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010. 14
- [5] Arthur J Baroody and Ann Dowker. *The development of arithmetic concepts and skills: Constructive adaptive expertise*. Routledge, 2013. 17
- [6] Charles H Bennett, Péter Gács, Ming Li, Paul MB Vitányi, and Wojciech H Zurek. Information distance. *Information Theory, IEEE Transactions on*, 44(4):1407–1423, 1998. 17
- [7] J. G. W. Bentley, P. G. Bishop, and M. van der Meulen. An empirical exploration of the difficulty function. In *Computer Safety, Reliability, and Security*, pages 60–71. Springer, 2004. 14
- [8] David Cornforth, David G Green, David Newth, and Michael Kirley. Do artificial ants march in step? ordered asynchronous processes and modularity in biological systems. In *Proceedings of the eighth international conference on Artificial life, MIT Press*, pages 28–32, 2003. 6
- [9] D. L. Dowe and J. Hernández-Orallo. IQ tests are not for machines, yet. *Intelligence*, 40(2):77–81, 2012. 27
- [10] D. L. Dowe and J. Hernández-Orallo. How universal can an intelligence test be? *Adaptive Behavior*, 22(1):51–69, 2014. 27
- [11] D. L. Dowe, J. Hernández-Orallo, and P. K. Das. Compression and intelligence: social environments and communication. In J. Schmidhuber, K.R. Thórisson, and M. Looks, editors, *Artificial General Intelligence*, volume 6830, pages 204–211. LNAI series, Springer, 2011. 27
- [12] Nazim Fates. A guided tour of asynchronous cellular automata. In *Cellular Automata and Discrete Complex Systems*, pages 15–30. Springer, 2013. 6

- [13] B. Goertzel. Toward a formal characterization of real-world general intelligence. In *The Third Conference on Artificial General Intelligence (AGI 2010)*, pages 19–24. Citeseer, 2010. 18
- [14] B. Goertzel, I. Arel, and M. Scheutz. Toward a roadmap for human-level artificial general intelligence: Embedding hlai systems in broad, approachable, physical or virtual contexts. *Artificial General Intelligence Roadmap Initiative*, 2009. 18, 27
- [15] J. Hernández-Orallo. Beyond the Turing Test. *J. Logic, Language & Information*, 9(4):447–466, 2000. 15, 25
- [16] J. Hernández-Orallo. A computational definition of ‘consilience’. *Philosophica*, 61:901–920, 2000. 15, 27
- [17] J. Hernández-Orallo. Computational measures of information gain and reinforcement in inference processes. *AI Communications*, 13(1):49–50, 2000. 2, 12, 15, 27
- [18] J. Hernández-Orallo. Constructive reinforcement learning. *International Journal of Intelligent Systems*, 15(3):241–264, 2000. 15, 27
- [19] J. Hernández-Orallo. On the computational measurement of intelligence factors. In A. Meystel, editor, *Performance metrics for intelligent systems workshop*, pages 1–8. National Institute of Standards and Technology, Gaithersburg, MD, U.S.A., 2000. 27
- [20] J. Hernández-Orallo. A (hopefully) non-biased universal environment class for measuring intelligence of biological and artificial systems. In M. Hutter et al., editor, *Artificial General Intelligence, 3rd Intl Conf*, pages 182–183. Atlantis Press, Extended report at <http://users.dsic.upv.es/proy/anynt/unbiased.pdf>, 2010. 27
- [21] J. Hernández-Orallo. On environment difficulty and discriminating power. *Autonomous Agents and Multi-Agent Systems*, pages 1–53, 2014. 27
- [22] J. Hernández-Orallo and D. L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508 – 1539, 2010. 9, 27
- [23] J. Hernández-Orallo and D. L. Dowe. On potential cognitive abilities in the machine kingdom. *Minds and Machines*, 23:179–210, 2013. 27
- [24] J. Hernández-Orallo, D. L. Dowe, S. España-Cubillo, M. V. Hernández-Lloreda, and J. Insa-Cabrera. On more realistic environment distributions for defining, evaluating and developing intelligence. In J. Schmidhuber, K.R. Thórisson, and M. Looks, editors, *Artificial General Intelligence*, volume 6830, pages 82–91. LNAI, Springer, 2011. 27
- [25] J. Hernández-Orallo, D. L. Dowe, and M. V. Hernández-Lloreda. Universal psychometrics: Measuring cognitive abilities in the machine kingdom. *Cognitive Systems Research*, 27:5074, 2014. 1, 3, 6, 18, 27
- [26] J. Hernández-Orallo and I. García-Varea. Explanatory and creative alternatives to the MDL principle. *Foundations of Science*, 5(2):185–207, 2000. 15, 27
- [27] J. Hernández-Orallo, J. Insa-Cabrera, D. L. Dowe, and B. Hibbard. Turing machines and recursive Turing Tests. In V. Muller and A. Ayesh, editors, *AISB/IACAP 2012 Symposium “Revisiting Turing and his Test”*, pages 28–33. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, 2012. 27
- [28] J. Hernández-Orallo, J. Insa-Cabrera, D. L. Dowe, and B. Hibbard. Turing Tests with Turing machines. In Andrei Voronkov, editor, *Turing-100*, volume 10, pages 140–156. EPiC Series, 2012. 27
- [29] J. Hernández-Orallo and N. Minaya-Collado. A formal definition of intelligence based on an intensional variant of Kolmogorov complexity. In *Proc. Intl Symposium of Engineering of Intelligent Systems (EIS’98)*, pages 146–163. ICSC Press, 1998. 15, 25
- [30] José Hernández-Orallo. Universal and cognitive notions of part. In Ferrer, L. et al. (eds.), *Proceedings of 4th Systems Science European Congress*, pages 711–722, 1999. 15, 27
- [31] José Hernández-Orallo. AI evaluation: past, present and future. *arXiv preprint arXiv:1408.6908*, 2014. 27
- [32] B. Hibbard. Bias and no free lunch in formal measures of intelligence. *Journal of Artificial General Intelligence*, 1(1):54–61, 2009. 27

- [33] M. Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(03):431–443, 2002. 20
- [34] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, 2005. 11, 20
- [35] J. Insa-Cabrera, J. L. Benacloch-Ayuso, and J. Hernández-Orallo. On measuring social intelligence: Experiments on competition and cooperation. In J. Bach, B. Goertzel, and M. Iklé, editors, *AGI*, volume 7716 of *Lecture Notes in Computer Science*, pages 126–135. Springer, 2012. 27
- [36] J. Insa-Cabrera, D. L. Dowe, S. España-Cubillo, M. V. Hernández-Lloreda, and J. Hernández-Orallo. Comparing humans and AI agents. In J. Schmidhuber, K.R. Thórisson, and M. Looks, editors, *Artificial General Intelligence*, volume 6830, pages 122–132. LNAI, Springer, 2011. 27
- [37] J. Insa-Cabrera, D. L. Dowe, and J. Hernández-Orallo. Evaluating a reinforcement learning algorithm with a general intelligence test. In J.A. Moreno J.A. Lozano, J.A. Gamez, editor, *Current Topics in Artificial Intelligence. CAEPIA 2011*. LNAI Series 7023, Springer, 2011. 27
- [38] J. Insa-Cabrera and J. Hernández-Orallo. Definition and properties to assess multi-agent environments as social intelligence tests. *arXiv preprint*, <http://arxiv.org/abs/1408.6350>, 2014. 27
- [39] J. Insa-Cabrera, J. Hernández-Orallo, D.L. Dowe, S. España, and M.V. Hernández-Lloreda. The anynt project intelligence test : Lambda - one. In V. Muller and A. Ayesh, editors, *AISB/IACAP 2012 Symposium “Revisiting Turing and his Test”*, pages 20–27. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, 2012. 27
- [40] Kelly Jaakkola, Wendi Fellner, Linda Erb, Mandy Rodriguez, and Emily Guarino. Understanding of the concept of numerically ‘less’ by bottlenose dolphins (*tursiops truncatus*). *Journal of Comparative Psychology*, 119(3):296, 2005. 3
- [41] S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, 2007. 27
- [42] S. Legg and J. Veness. An approximation of the universal intelligence measure. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 236–249. Springer, 2013. 27
- [43] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973. 2, 11, 13
- [44] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications (3rd ed.)*. Springer-Verlag, 2008. 2, 11, 13, 20, 21
- [45] Peng Liu and Zhizhong Li. Task complexity: A review and conceptualization framework. *International Journal of Industrial Ergonomics*, 42(6):553–568, 2012. 9
- [46] H. Masum, S. Christensen, and F. Oppacher. The Turing ratio: Metrics for open-ended tasks. In *GECCO*, pages 973–980. Citeseer, 2002. 18, 27
- [47] J. E. Mayfield. Minimal history, a theory of plausible explanation. *Complexity*, 12(4):48–53, 2007. 2, 19, 27
- [48] Bonnie M Perdue, Catherine F Talbot, Adam M Stone, and Michael J Beran. Putting the elephant back in the herd: elephant relative quantity judgments match those of other species. *Animal cognition*, 15(5):955–961, 2012. 3
- [49] B. Rohrer. Accelerating progress in artificial general intelligence: Choosing a benchmark for natural world interaction. *J. of Artificial General Intelligence*, 2(1):1–28, 2010. 18, 27
- [50] P. Sanghi and D. L. Dowe. A computer program capable of passing IQ tests. In *4th Intl. Conf. on Cognitive Science (ICCS’03), Sydney*, pages 570–575, 2003. 15
- [51] T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 5(6):4650, 2010. 20
- [52] J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004. 20
- [53] J. Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence*, pages 199–226. Springer, 2007. 20

- [54] Christian R Shelton and Gianfranco Ciardo. Tutorial on structured continuous-time Markov processes. *Journal of Artificial Intelligence Research*, pages 725–778, 2014. 5, 6
- [55] A.L. Strehl, L. Li, E. Wiewiora, J. Langford, and M.L. Littman. PAC model-free reinforcement learning. In *Proc. of the 23rd Intl Conf on Machine learning*, ICML '06, pages 881–888, New York, 2006. 27
- [56] L. L. Thurstone and T. G. Thurstone. *American Council on Education for college freshmen: Manual of Instructions*. Cooperative Test Division, Educational Testing Service, 1947. 15
- [57] A. M. Turing. Intelligent machinery, a heretical theory. *The Turing Test: Verbal Behavior as the Hallmark of Intelligence*, page 105, 1948. 2, 20
- [58] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. 20
- [59] Leslie Valiant. *Probably Approximately Correct: Nature’s Algorithms for Learning and Prospering in a Complex World*. Basic Books, 2013. 20
- [60] J. Veness, K.S. Ng, M. Hutter, and D. Silver. A Monte Carlo AIXI Approximation. *Journal of Artificial Intelligence Research, JAIR*, 40:95–142, 2011. 8, 12
- [61] Paul MB Vitányi, Frank J Balbach, Rudi L Cilibrasi, and Ming Li. Normalized information distance. In *Information theory and statistical learning*, pages 45–82. Springer, 2009. 17
- [62] Jennifer Vonk and Michael J Beran. Bears ‘count’ too: quantity estimation and comparison in black bears, *ursus americanus*. *Animal behaviour*, 84(1):231–238, 2012. 3
- [63] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992. 8